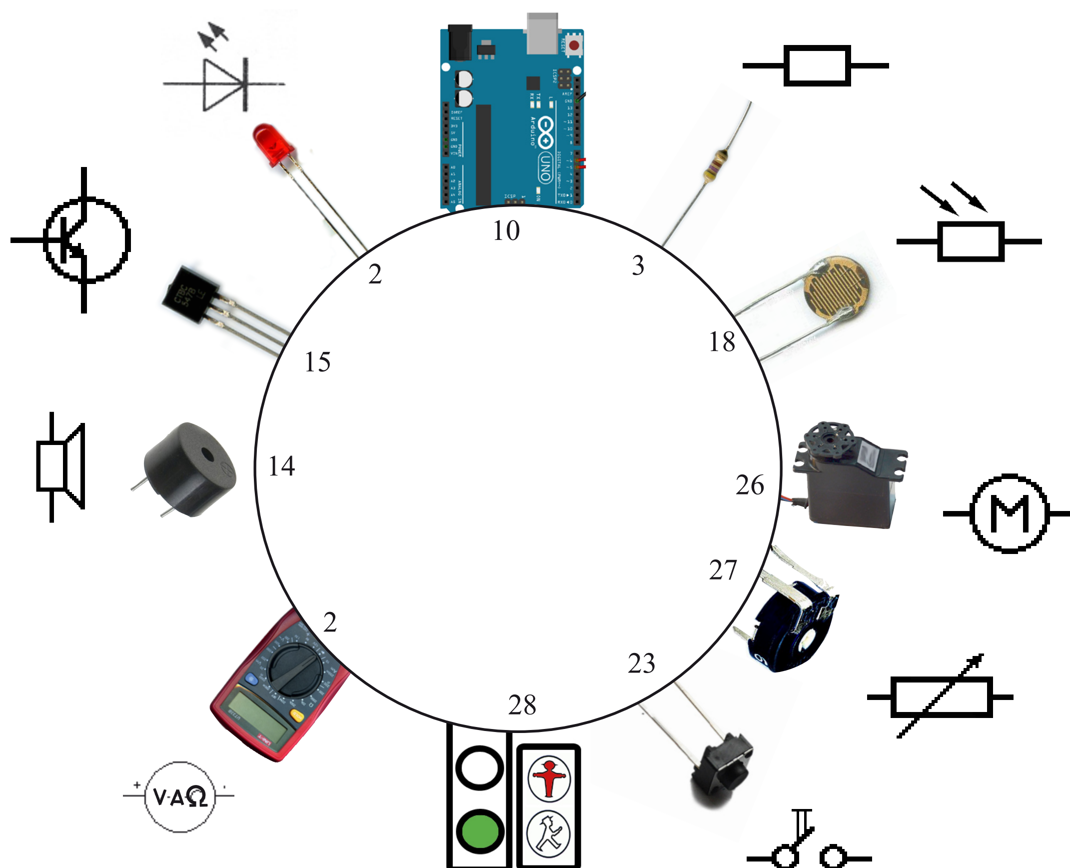
Arduino Nr.	HA-Kasten	Reserve	Name:

NwT 9: Programmieren mit dem Arduino

Version 2020 Bach und Weitbrecht



weitere Themen
16 Sieben-Segment-Anzeige
23 PWM
16 IR-Fernbedienung, CNY 70
25 H-Brücke, Bibliotheken
32 Probleme
34 Befehlsübersicht
36 Programmierfehler

Kabelfarben und ihre übliche Bedeutung

schwarz	GND	egal woher
rot	5V	Dauerplus vom Arduino
orange		Fremde (positive) Betriebsspannung (Batterie, Trafo,...)
gelb	Pin 2-13	Ausgangssignale /Datensignal Servo zwischen 0V und 5V (PWM)
grün	A0, Pins	Eingangssignale Sensor/Taster

Aufgaben in purpurner Schrift
Lernziele und Anmerkungen in türkis
Spezialwissen für Nerds in grün

Wiederholung

Das Multimeter kann Widerstände und Spannungen messen sowie die Leitfähigkeit prüfen.

In NwT kommen mind. sechs verschiedene Multimetertypen (=Vielmesser) zum Einsatz. Deswegen hier die schematische Darstellung eines hypothetischen Multimeters mit den Optionen die immer ähnlich sind und einigen Besonderheiten, die evtl. nützlich sind, z.B. zur Überprüfung von Bauteilen.

Hausaufgabe 2.1: Bringt eine Mischpackung „Haribo" mit, die man nach zwei sinnvollen Kriterien (z.B. Farbe & Form) sortieren kann. Zum Beispiel Colorado, Fantasia oder Tropifrutti oder zwei Packungen Bio-Gummibärchen.

Ganz wichtig!
Am Ende den Drehschalter auf „OFF" stellen bzw. mit dem roten Knopf ausschalten. Sonst sind die Batterien bis zur nächsten Stunde erschöpft.

Batterie-Warnung - Neue Batterien einsetzen.

Beim Drücken der „HOLD"-Taste wird der aktuelle Messwert gespeichert und im Display mit einem „H" in der linken oberen Ecke angezeigt. Falls die roten Multimeter „nicht funktionieren" liegt das oft an einer ungewollt gedrückten „HOLD"-Taste.

Widerstand messen immer ohne Spannung, da das Multimeter selbst eine Messspannung an die Messspitzen anlegt um den Widerstand wahrscheinlichmit einem Spannungsteiler zu messen.
Beginne bei der Messung mit dem kleinsten Messbereich „200" und erhöhe ihn, bis sonnvolle Messwerte ablesbar sind.

Stromstärke messen

Dioden-Test = Durchgangsprüfung auf Leitfähigkeit. Bei den blauen Multimetern mit Signalton.

Rote Multimeter mit Batterietest (mit internem Verbraucher)

Blaue und schwarze Multimeter mit Transistor-Test-Buchse. Unsere BC547 sind PNP-Transistoren, der Aufbau der anderen Transisitoren muss dem Datenlatt entnommen werden.

Schwarze Multimeter mit Kondensatortestbuchsen

Die Anzeigen .1 oder OL (Over Limit) zeigen an, dass du den Messbereich anpassen musst.

Hintergrundbeleuchtung bitte immer ausschalten um die Batterielaufzeiten zu verlängern.

Gleichstrom-Spannungsmessung für die Elektronikaufbauten in NwT sind „20V" die richtige Einstellung.

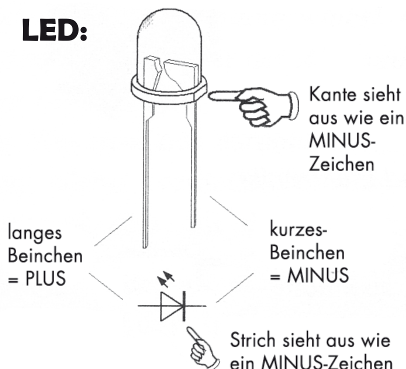
VERBOTEN!
Wer Steckdosen ausmisst und überlebt erhält Versuchsverbot! (Wechselstrommessbereich)

BItte so anschließen:

Messspitzen nicht in Mitschüler*innen stecken.

.1 OL
HOLD
OFF
OHM Ω
20M
200k
20k
2k
200
V ⎓
20
200
600
V ~
A ⎓
12V 9V 1,5V
CAP
10A
V Ω mA
COM
B E E C B E C E
NPN PNP

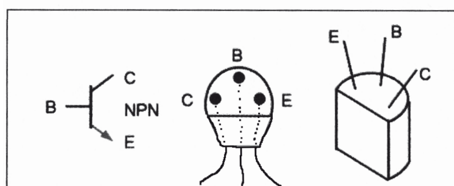
LED:



Transistor:

Zur Funktion eines Transistors sind einige grundlegende Voraussetzungen erforderlich. Die stark positive Betriebsspannung (+ U Batt.) wird über einen Strombegrenzungswiderstand R C (Kollektorwiderstand) dem Kollektoranschluß zugeführt.

Der Emitteranschluß wird mit dem Minuspol (0 V) der Gleichspannungsquelle verbunden. Die Basis erhält ebenfalls über einen Widerstand R B (Basisvorwiderstand) eine wesentlich geringere Gleichspannung von ca. 0,7 V, die sog. U BE.



manchmal auch mit - gekennzeichnet

z.B. Autobatterie: 12 V (+)

Starker Strom rein (Collector)
C

Starker Strom raus (Emitter)
E

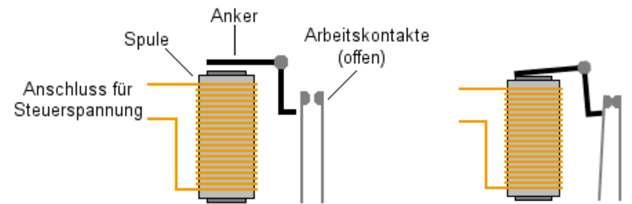
z.B. Autobatterie: 12 V (-)

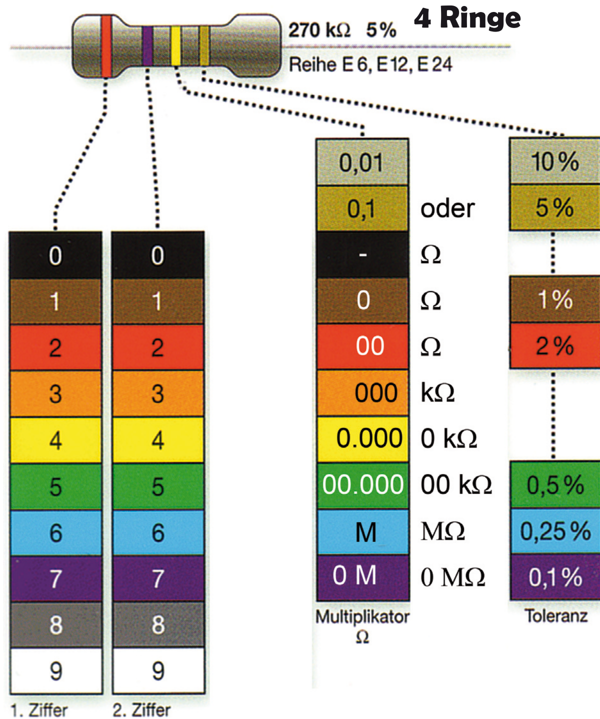
Kleiner Steuerstrom (> ~0,7 V) steuert den Durchfluss des starken Stromes.
Der Transistor wirkt hier als Stromventil.
(Basis)
B

z.B. Arduino-Signal 4,5 V (+)

Relais

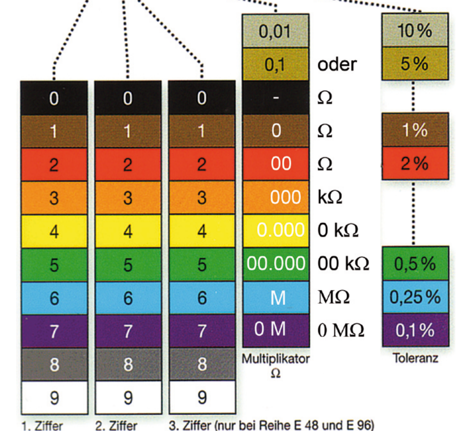
Ein Relais schaltet eine große Spannung/Stromstärke frei wenn eine kleine Steuerspannung angelegt wird. Ein kleiner Elektromagnet zieht einen Anker heran und dieser drückt zwei Arbeitskontakte (großer Strom) zusammen. Relais packen problemlos richtig große Spannungen - Transistoren werden dabei zu heiß.

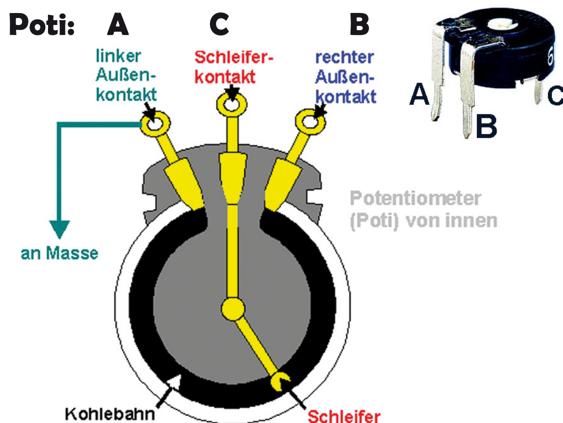


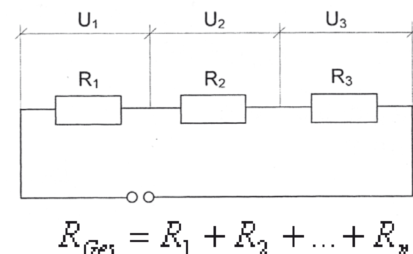


Widerstände: Farbcodierung

Farbige Ringe codieren den Wert des Widerstands. Der Code startet am „ersten Ring“ - dieser sitzt immer ganz außen. Der letzte Ring der Toleranzring ist meist mit etwas Abstand abgesetzt und er hat (bei den Standardformen) meist die besonderen Farben silber und gold.



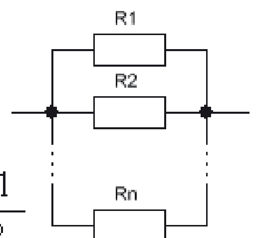




$$R_{Ges} = R_1 + R_2 + \ldots + R_n$$

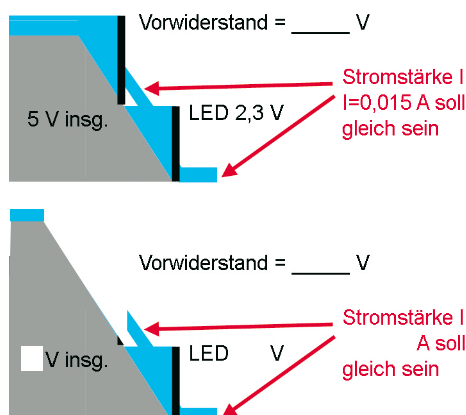
Reihenschaltung: Stromstärke I ist überall gleich, Spannung U wird aufgeteilt (mehr Widerstand = mehr Spannung), berechne $U_i = R_i \cdot I_{ges}$

Parallelschaltung: Spannung U ist überall gleich. Stromstärke I ist aufgeteilt, (mehr Widerstand = weniger Strom). $I_i = U_{ges} / R_i$



$$\frac{1}{R_{Ges}} = \frac{1}{R_1} + \frac{1}{R_2} + \ldots + \frac{1}{R_n}$$

Vorwiderstand berechnen



Aus dem Datenblatt kann man ablesen:
Unsere LEDs halten eine Stromstärke von 15 mA = 0,015 A (Ampère) aus bei einer Spannung von 2,3 V. Da der Microcontroller aber eine Spannung von 5 V liefert, müssen die überschüssigen Volt von einem Schutz- oder Vorwiderstand abgefangen werden, der auch gleichzeitig die Stromstärke auf die empfohlenen 0,015 A begrenzt.

Von den 5 V fallen 2,3 V an der LED ab, also bleiben für den Vorwiderstand noch 2,7 V übrig. Bei der gewünschten Stromstärke von 0,015 A sollte der Vorwiderstand (R = U/I) also 2,7 V / 0,015 A = 180 Ω (Ohm) betragen.
Falls dieser Widerstand gerade nicht vorhanden ist, verwenden wir den nächstgrößeren, also 220 Ω

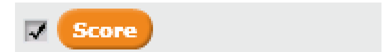
Programmstrukturen

Scratch im Vergleich
Es ist schon zwei Jahre her, damals ....
analoger C++ Befehl

Variablen

werden global (für die gesamte Programmlaufzeit) oder lokal (nur für eine Schleife/eine Funktion) definiert.
Mit der Definition wird ein entsprechender Speicherplatz auf dem Mikrocontroller reserviert.

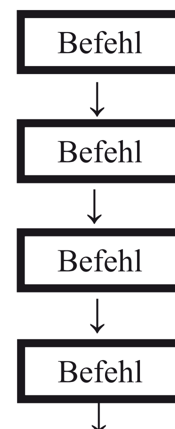
Variablen sind Datenspeicher. Sie werden mit einem Namen benannt (z.B. „a") und beinhalten Werte, Zahlen oder Buchstaben



int score;

int (integer) reserviert zwei Byte Speicherplatz (1 Byte=8Bit=8 Ziffern) für eine Zahl

Ein Rechner kann immer nur eine Anweisung nach dem anderen innerhalb einer
Befehlskette
abarbeiten (Ausnahme Dual-Core-Chips, Parallelrechner, u.ä.).





score=0;

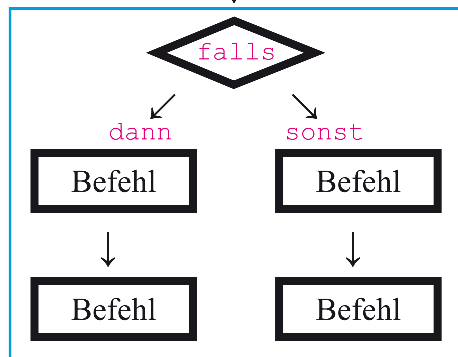


score=score+1;
oder score++;



Serial.print (score);

Wenn der Rechner je nach Eingabe oder Senorwert verschiedene Dinge tun soll (z.B. wenn „dunkel", dann „Licht an", sonst „Licht aus") verwendet man **Verzweigungen.**



if (score==2) { ...Anweisungen... }
else { ...andere Anweisungen... }

Wenn es nur eine „falls...dann..."-Abfrage ist nennt man das eine
Bedingung

eigentlich ist eine Bedingung auch eine Verzweigung, denn der µC soll („else") einfach weiter machen wenn die Bedingung nicht erfüllt ist. Aber wenn es der Lehrplan so will...



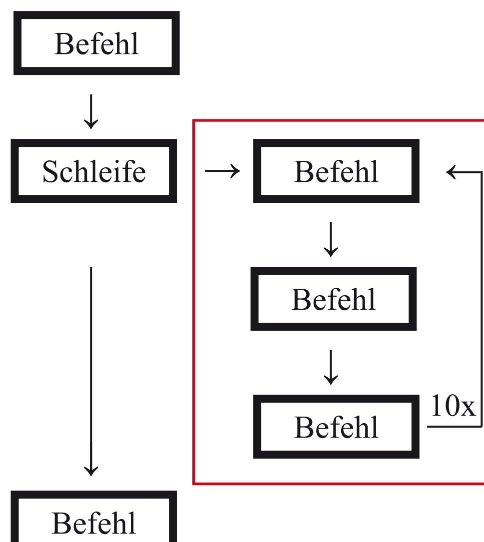
if (score>2) { ...Anweisungen... }

Schleifen

werden verwendet, wenn der Rechner eine Befehlskette mehrmals wiederholen soll.

Eine For-Schleife wird benutzt wenn die Zahl der Wiederholungen bekannt ist. (Roboter soll drei mal winken).

Weitere Schleifen arbeiten mit Abbruchbedingungen oder Mindestwerten (solange noch zu kalt bitte heizen). (while-Schleife).





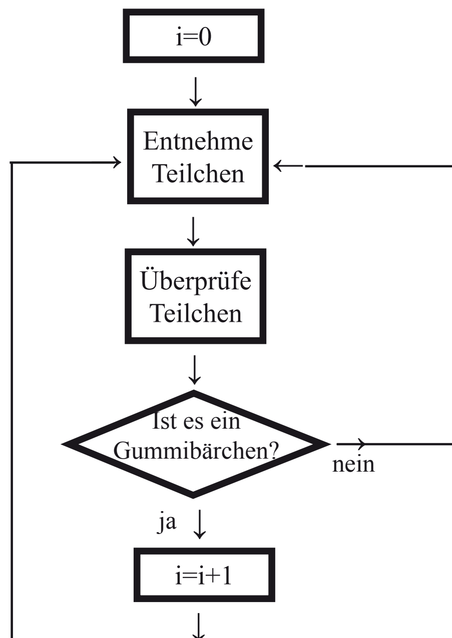
for (int i=1, i<10, i++)
{ ...Anweisungen... }

integer i wird hier nur lokal für die Dauer der Schleife (zehn Durchgänge) reserviert und dann wird der Speicherplatz wieder freigegeben.

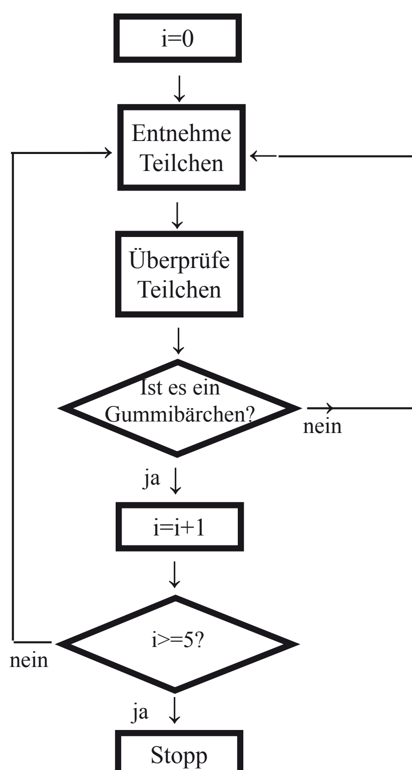
Programme

Aus dem NwT-Lehrplan:
Anweisungen, globale und lokale Variablen, Rechenoperationen, Bedingungen, Verzweigungen, Schleifen und Unterprogramme

Aufgabe 5.1:
a) Was macht das Programm?
b) Welche der Elemente „Variable, Befehlskette, Verzweigung, Bedingung und Schleife“ sind in diesem Programm enthalten? Ordne zu!



Aufgabe 5.2
Was macht dieses erweiterte Programm?



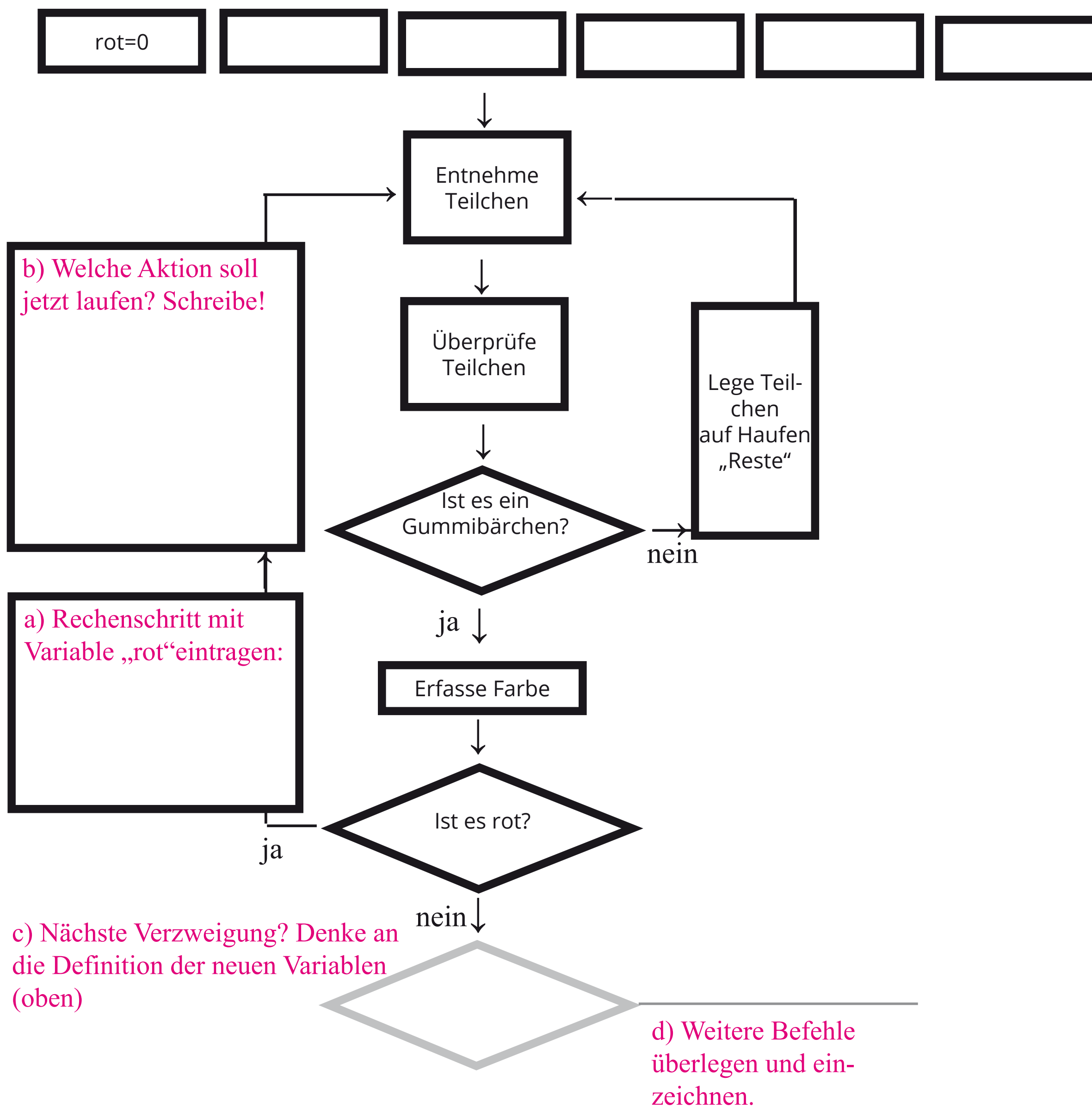
Hausaufgabe war: Bringt eine Mischpackung „Haribo“ mit die man nach zwei sinnvollen Kriterien (z.B. Farbe & Form) sortieren kann. Zum Beispiel Colorado, Fantasia oder Tropifrutti oder zwei Packungen Bio-Gummibärchen.

Lösung zu 5.1:

Lösung zu 5.2

Aufgabe 5.3
Wie müsste man das Programm erweitern, um auch die „Nicht-Gummibärchen“ zu „erfassen“? Ergänze das Programm links oder zeichne das Struktogramm des neuen Programms auf ein Extrablatt und klebe es hier ein.

Aufgabe 6.1:
Das folgende Programm soll Gummibärchen der Farbe nach zählen UND sortieren. Fülle das Struktogramm aus a),b) und c) und ergänze weitere Felder für mindestens eine/zwei/drei weitere Farben (für alle Farben mit zusätzlich eingeklebten Blättern).

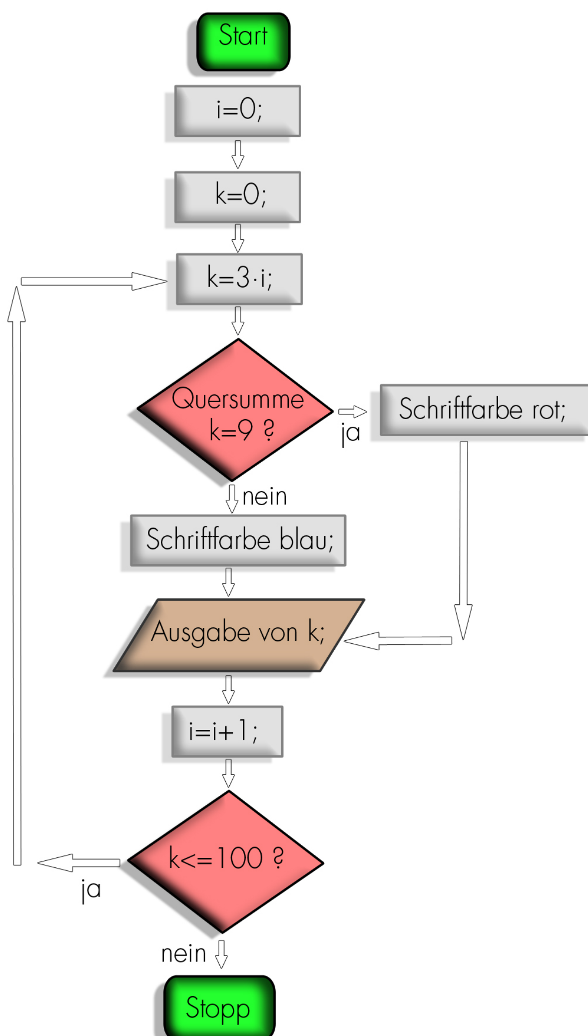


Klebefläche

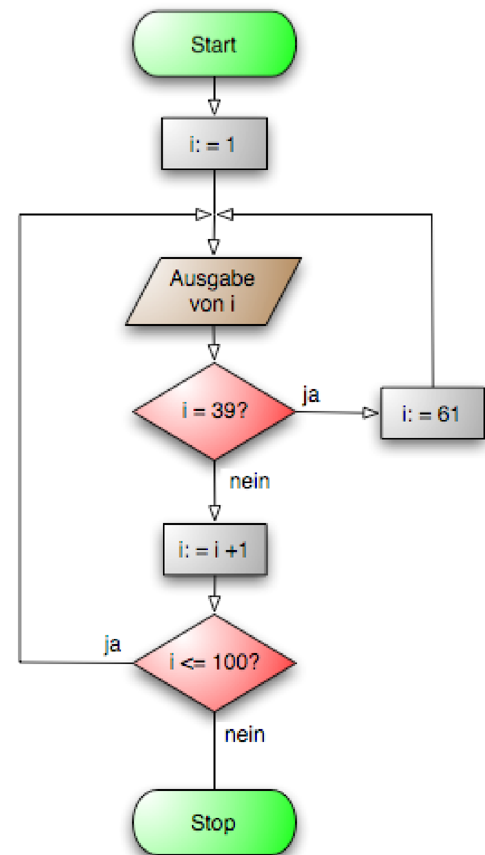
Lernziel:
Unterscheidung zwischen intern berechneten und extern angezeigten Variablen. Was ein Programm rechnet und was es ausgibt sind zwei paar Stiefel.

Aufgabe 7.1:
Das Programm soll bestimmte Zahlen zwischen 0 und 100 ausgeben.
a) Welche Zahlen sind das?

b) Das Programm macht nicht ganz was es soll. Finde den Fehler! Dabei hilft dir Teilaufgabe c)
c) Welche drei Zahlen gibt das Programm als letzte aus? Bist du wirklich sicher???
d) Welchen Befehl müsste man wie abwandeln, damit alles richtig klappt?



Aufgabe 7.2:
Was rechnet das (völlig sinnlose) Programm?



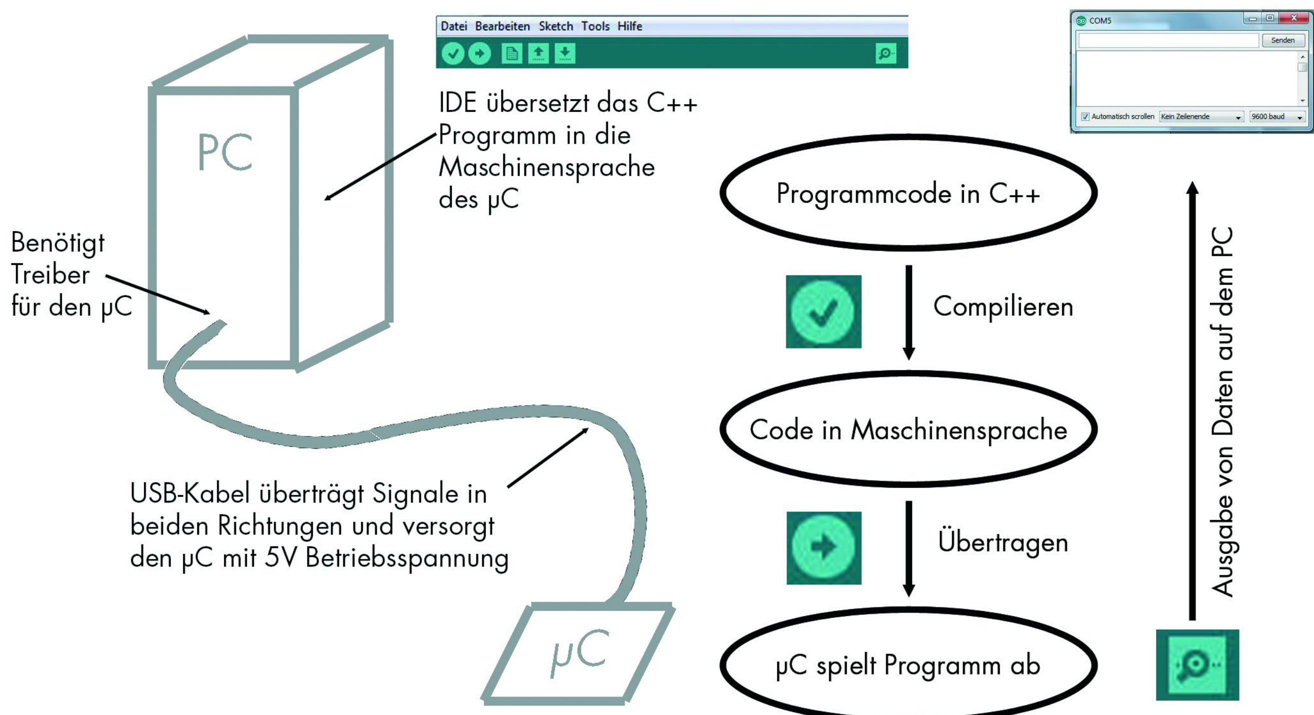
Aufgabe 7.3 Schwere KA-Frage
Und was GENAU gibt das Programm aus?

Lernziele:
Grundprinzip nachvollziehen
drei Schritte aufzählen bis der µC das tut was man will

Programmieren mit dem Arduino Mikrocontroller

Material und grundsätzlicher Ablauf

1) Material: Mikrocontroller (µC) Arduino UNO R3, USB-Kabel, elektronische Bauteile
2) USB-Kabel vorne am PC einstecken
3) Arduino IDE (integrated development environment) starten
4) Programm in der Programmiersprache C++ laden oder schreiben
5) µC mit einer Schaltung aus elektronischen Bauteilen bestücken
6) Fertiges Programm compilieren, d.h. in Maschinensprache übersetzen und auf den µC überspielen



Klartext:

Manchmal glauben Schüler*innen, sie wären besonders schlau und kopieren sich den fertigen Programmcode aus dem Internet oder über den Tauschordner von Mitschüler*innen.

Das ist natürlich auch eine Fertigkeit, allerdings nicht die, die in der Klassenarbeit abgefragt wird, dort müsst ihr selbst programmieren (und evtl. auch stecken). Wer das vorher nicht richtig geübt hat, wird grandios scheitern.

Jede und jeder soll selbst programmieren - nur vom Zuschauen kann man das nicht gut lernen. Ihr bekommt genug Zeit (evtl. dann halt noch Zuhause) und Hilfe um es selbst verstehen zu können.

Material: UNO R3, Steckplatine, Box Arduino Training (für alle Programmieraufgaben)

Für Hausaufgaben: UNO R3 Kit im Kasten

Arduino 1

Hausaufgabe 9.1 Installation der IDE daheim!

1. http://www.arduino.cc
   a. Download
   b. Eigenes Betriebssystem auswählen
   c. Zip-Archiv entpacken
2. arduino.exe starten und programmieren
3. Arduino Uno und PC mit dem USB-Kabel verbinden und unter Windows dann den passenden Treiber aus dem Unterverzeichnis /driver/ aus dem entpackten Zip-Archiv installieren (lassen).
4. fertig

Am KvFG gibt es inzwischen ein nettes Sammelsurium diverser UNO R3 von unterschiedlichen Herstellern mit unterschiedlichen USB-Anschlüssen - aber alle können das Gleiche.

Du bekommst für die Dauer dieser Lerneinheit ZWEI Arduinos zur Verfügung gestellt: Ein komplettes Kit für Zuhause. und einen in einer Box in der Schule mit den Komponenten die ihr für die Aufgaben in diesem Heft benötigt.

Aufgabe 9.1: Notiere dir die Nummer der Schulbox auf der ersten Seite.

Der Arduino und die benötigten Bauteile bleiben immer in deiner Schachtel/Box.

Aufgabe 9.2: Beschrifte die Box mit deinem Namen, Klasse und Schuljahr.

Aufgabe 9.3: Besorge dir einen Rechnerplatz / einen Laptop wenn es nicht genügend Rechner für alle gibt, dürft ihr auch ausnahmsweise zu zweit arbeiten.

Aufgabe 9.3: Überprüfe dein Kit für Zuhause auf Vollständigkeit und trage deinen Namen, Klasse und Schuljahr in den Leihzettel ein. Wenn im Folgejahr entdeckt wird, dass Teile fehlen, musst du sie prinzipiell bezahlen.

Wann musst du sie nicht bezahlen: Wenn sie defekt waren und du das -spätestens bei der Rückgabe - mitgeteilt hast und von der*dem Lehrer*in Ersatz bekommen hast. Bei grober Gewalt (The Hulk!) zahlt du aber selbst!

Hausaufgaben mit dem Arduino?

Du benötigst einen Rechner mit Arduino IDE, wenn das Probleme machen sollte, wende dich bitte vertrauensvoll an deinen Fachlehrer, mal sehen ob wir da was machen können...

Man kann für das Programmieren und Stecken viel Zeit aufwenden. Viele Schüler*innen haben voll Spaß dabei und bauen zusätzliche eigene Projekte - ist selbstverständlich erlaubt. Wenn es aber zu einer allzu langen Quälerei wird, lasst es von einem Erziehungsberechtigten bestätigen und ihr bekommt (fast) keinen Ärger.

Zuhause fehlt dir die schnelle Hilfe durch den Lehrer, deswegen beachte insbesondere bei

Fehlermeldungen die Rückseite

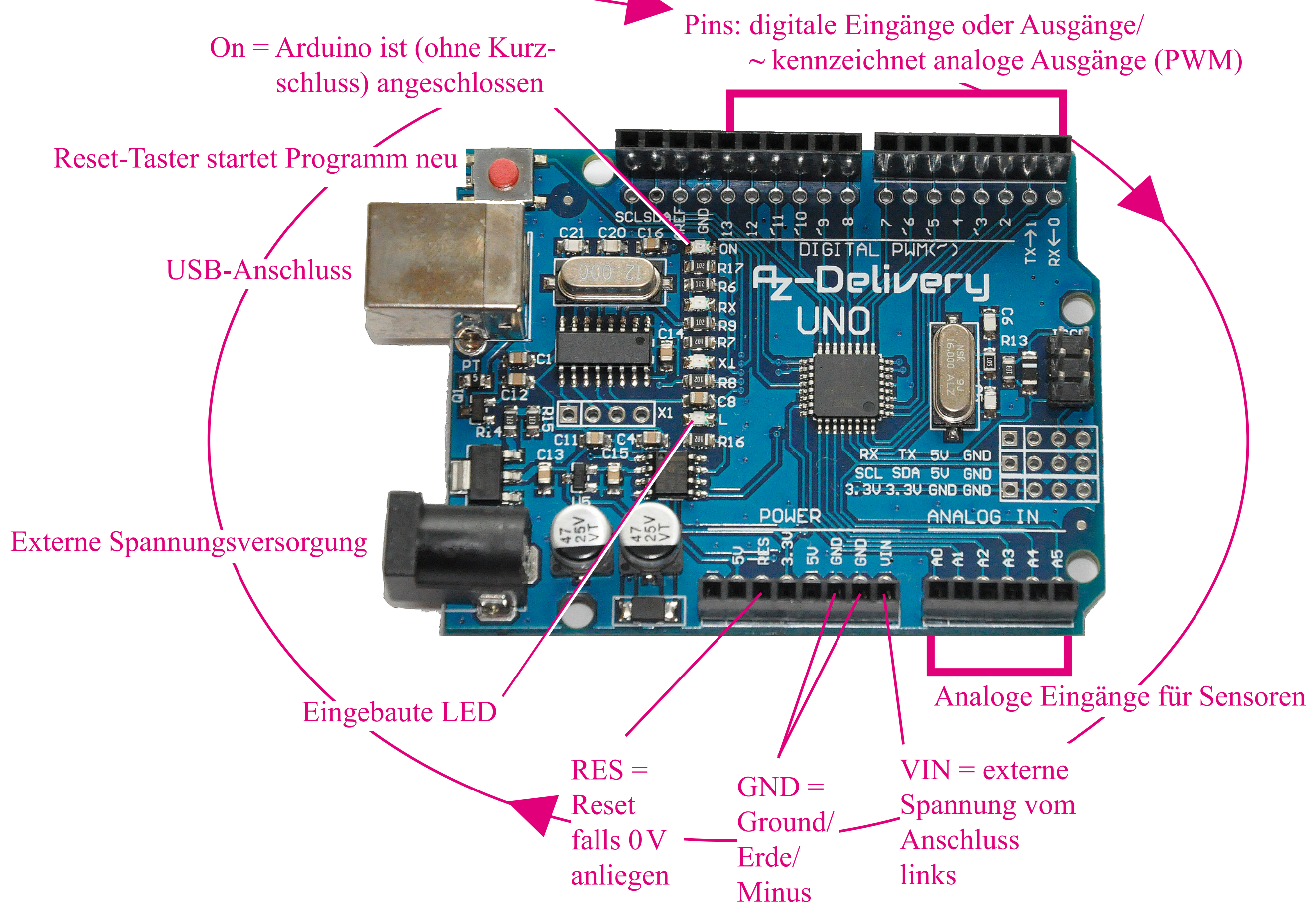
bei Problemen insgesamt Seite 32.

Zur Übertragung deiner Programme zwischen Schul-Arduino und HA-Arduino bietet sich die Nutzung der HORDE-Plattform im Schulnetzwerk an, damit kannst du von Zuhause aus auf dein Schulverzeichnis zugreifen. Anleitung steht im Schulplaner unter Netzwerk.

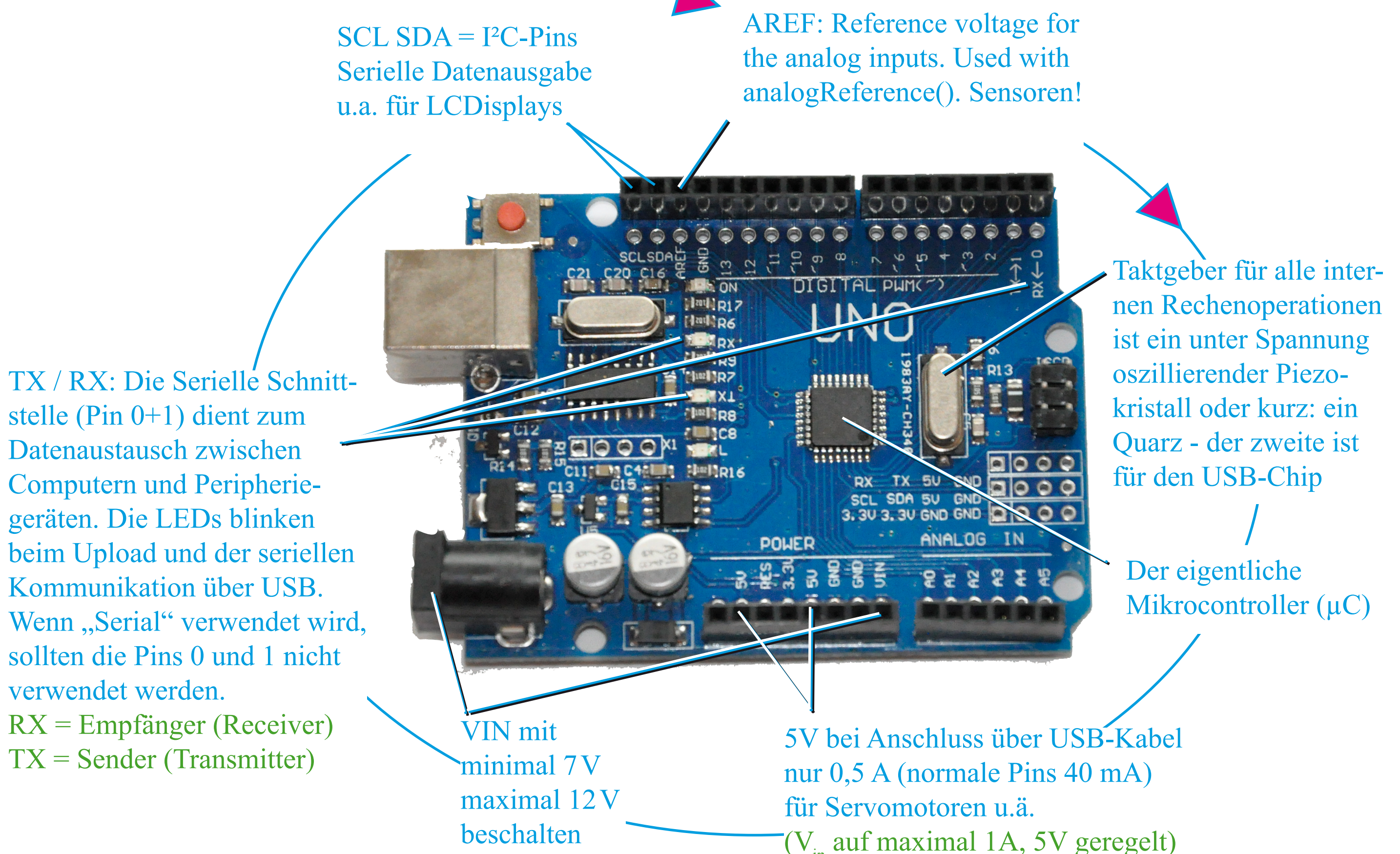
Weitere Hinweise:

Lernziele:
Alle Anschlüsse, Begriffe und Bauteile auf Seite 10 wissen.

Lernziel 1 Grundwissen



Lernziel 2 Detailwissen



Bonus Nerdwissen und Hausaufgaben Arduino

Pins mit Sonderfunktionen

Eine Besonderheit stellt der Digital-Pin 13 dar: Dort ist der Ausgabe-Strom auf nur 20 mA begrenzt, so dass dort eine LED direkt (ohne Vorwiderstand) angeschlossen werden kann (direkt neben Pin 13 ist ein GND-Pin, so dass dafür nicht einmal eine Steckplatine nötig ist).

Die Stromstärke als Output ist bei Pin 0-12 auf 40 mA begrenzt; gegebenenfalls (zu viele LEDs an einem Ausgang) wird die Spannung der Pins (eigentlich 5 V) automatisch herab geregelt, um diese Begrenzung zu erreichen.

Die digitalen Pins 0 bis 13 können ebenfalls als Sensor-Eingänge festgelegt werden: Eine anliegende Spannung von >2,5 V wird als HIGH (Zahlenwert 1), eine niedrigere Spannung als LOW (Zahlenwert 0) interpretiert.

SCK/MISO/MOSI/SS: Das SPI „Serial Peripheral Interface“ ist ein Bus-System an dem theoretisch beliebig viele Gerät/Sensoren angeschlossen werden können, wobei es jedoch immer genau einen Master geben muss. Die Vorteile von SPI sind eine höhere Datenrate und separate Ein- und Ausgänge, so dass gleichzeitig gesendet und empfangen werden kann. Der Nachteil ist dafür dass eine zusätzliche Leitung pro Bauelement/Gerät benötigt wird. SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library

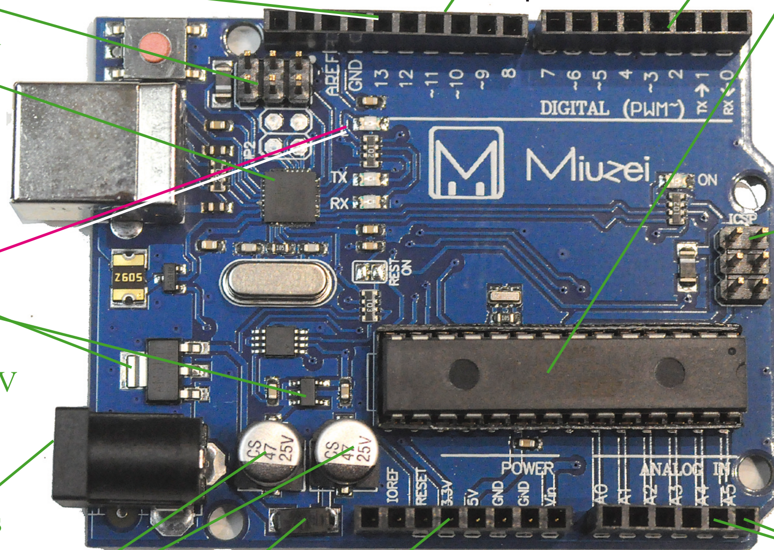
External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details. Z.B. Steuerung über PC.

Der eigentliche Mikrocontroller (µC) (Atmega 16U2) in IC-Bauweise. Wie der Original-Arduino ist er aufgesteckt und kann bei Defekt ausgetauscht werden.

Mit ICSP und diesen 6 Pins könnte man den USB-Interface-Chip neu programmieren. Die vier Buchsen sind freie GPIO-Pins des gleichen Chips.

Eingebaute LED

Spannungsbegrenzer 5,0 V mit Kühllasche und der kleine für 3,3 V



ICSP zur Programmierung des µC mit einem Bootloader

„5,5 mm außen Minus 2,1 mm innen“ Plus-Hohlstecker

Kondensatoren (für Spannungsregler?)

Diode als Verpolungsschutz

3,3 V für besondere Ansteckteile mit maximal 50 mA

A0 bis A5: durch einen eingebauten Analog-Digital-Wandler werden die gemessenen Spannungswerte auf einem Zahlenbereich von 0 (keine Spannung) bis 1023 (maximale Spannung, also 5 V) abgebildet.

SDA / SCL: Mit I²C „Inter-Integrated Circuit“ (TWI = „Two Wire Interface“) (Pins A4+A5) kann man ein ganzes Netzwerk an integrierten Schaltungen mit nur zwei I/O Pins und einer einfachen Software kontrollieren. Die Nachteile sind, dass die Datenraten geringer sind als bei SPI und dass die Daten nur jeweils in eine Richtung fließen können. Für einen leichteren Zugriff wurden die beiden Anschlüsse A4 = SDA und A5 = SCL (bei den anderen Bauformen auch beschriftet) nochmal links oben rausgeführt. SDA und SDL sind dort keine zusätzlichen Anschlüsse sondern sie sind direkt mit A4 und A5 leitend verbunden. Support TWI communication using the Wire library.

Grundaufbau eines C++-Programms (in der IDE als Sketch bezeichnet)

```
int led = 13;

void setup() {
  pinMode(led, OUTPUT);
}

void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Definitionsbereich

Setup-Bereich

Loop = Hauptprogramm

Die Arduine Entwicklungsumgebung - kurz IDE (integrated development environment) starten. Gib dazu in das Feld links unten einfach die Anfangsbuchstaben „Ard“ ein und das Betriebssystem findet das bläuliche Symbol für dich. Draufklicken und los geht es!

Die Entwickler*innen der IDE haben eine bestimmte Programmstruktur festgelegt.

Globale Variablendefinitionen und die Einbindung fertiger Befehlssammlungen (z.B. „servo.h“) erfolgen in den ersten Zeilen des Programms - dem Definitionsbereich. Globale Variablen gelten für das gesamte Programm inklusive aller Unterprogramme = Funktionen.

Der Bereich der in dem die Parameter für das Arduino-Board festgelegt werden, wurde „setup“ getauft. Im Setup werden Ein- und Ausgänge, die Einstellungen für die Kommunikation mit dem Computer u.ä. festgelegt. Dieser Bereich wird vom Arduino nur einmal durchlaufen.

Der dritte Bereich wird immer wieder wiederholt - in einer Endlosschleife, deswegen haben sie ihm den Namen „loop“ = Schleife gegeben. Hier stehen alle Befehle drin, die der Reihe nach abgearbeitet werden sollen. Die Befehle stehen immer zwischen den beiden geschweiften Klammern. Am Ende angelangt beginnt der Arduino wieder beim ersten Befehl innerhalb der geschweiften Klammern.

Die geschweiften Klammern fassen immer mehrere Befehle zu einer Einheit zusammen, das gilt zum Beispiel auch für if, else und für Funktionen.

Jeder Befehl wird mit einem besonderen Zeichen abgeschlossen, damit die IDE weiß, dass der Befehl hier zu Ende ist. Die Entwickler*innen haben sch dafür den Strichpunkt „ ; “ herausgesucht.

Hast du schon bemerkt, dass einige Programmteile beim Tippen automatisch farbig eingefärbt werden? Befehle in Hellbraun und Parameter in Blau. So kannst du direkt überprüfen, ob/dass du dich nicht vertippt hast.

Compilieren



Upload

wenn du später mal einfach nur Zahlen änderst oder so kannst du auch direkt hochladen klicken, es wird dann automatisch compiliert und hochgeladen

C++ Code

Vertiefung Hausaufgabe 13.4
Stecke und programmiere eine Autoampel.

Aufgabe 13.1

a) Starte die Arduino-IDE! (links unten draufklicken, „ard“ in der Programmsuche eingeben, blaues Symbol klicken)

b) Tippe das Beispielprogramm im Kasten links ab. Erinnere dich: Woran sind die drei Bereiche unterscheidbar?

c) Verbinde µC und PC mit dem USB-Kabel.

d) Compiliere das Programm, indem du auf den Haken klickst. Probleme? Lösung vgl. Rückseite.

e) Übertrage das Programm (upload) indem du auf den Pfeil klickst. Probleme? Lösung vgl. S.32.

f) Überprüfe: Blinkt die eingebaute LED?

Aufgabe 13.2

a) Speichere das Programm (erlaube die Einrichtung eines neuen Ordners auf deinem Homeverzeichnis).

b) Lass die LED doppelt so schnell blinken!

c) Schließe eine externe LED nach der Anleitung auf dem Foto an. Der Vorwiderstand sollte 220 Ohm betragen. Farbcode auf Seite 3.

d) Hausaufgabe: Baue eine zweite LED am Pin 11 ein und lass beide in einem netten Rhythmus blinken.

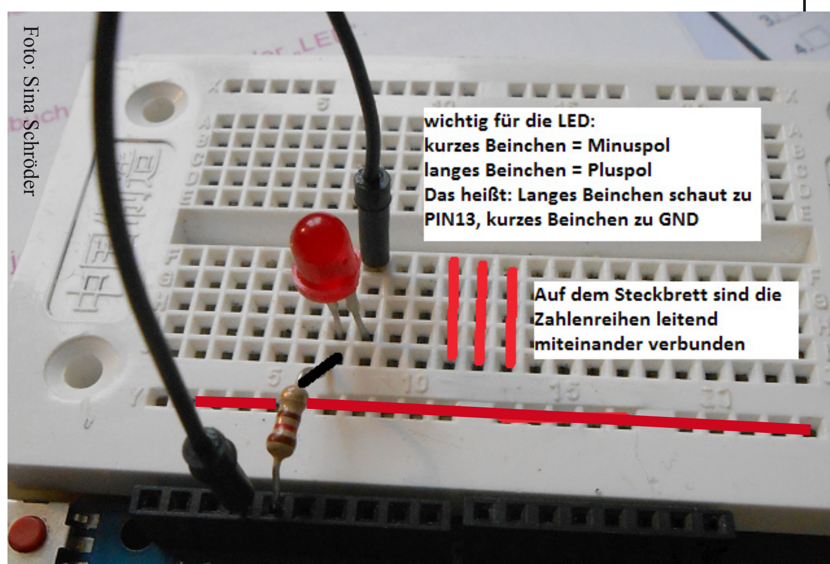


Foto: Sina Schröder

Aufgabe 13.3

a) Finde heraus, wie kurz die Pause sein darf, so dass du sie gerade noch als Pause erkennen kannst. Entspricht der Aufgabe: Mach die LED so schnell an und aus, dass dein Auge/Gehirn es als kontinuierliches Leuchten wahrnimmt.

FL: **13.3** a

b) Lass die LED „SOS“ blinken! (Recherchiere den Morsecode für SOS!)

FL: **13.3** b

c) Verwende eine for-Schleife um die LED immer schneller blinken zu lassen.

for für definierte Anzahl an Durchläufen:

```
for (int i=0; i<20; i++)
        // deklariert ‚i‘,
        // teste ob weniger als 20,
        // Erhöhe i um 1
{
        //hier die Befehle tippen
}
```

FL: **13.3** c

Anmerkung: Aller `Programmcode` ist in `Schreibmaschinenschrift` abgedruckt, die hinterlegte Farbe zeigt, in welchen Bereich die Befehle eingegeben werden müssen.

Bauteile schonend stecken

Die Beinchen der LED müssen bei der im Bild gezeigten Steckweise nicht verbogen werden.

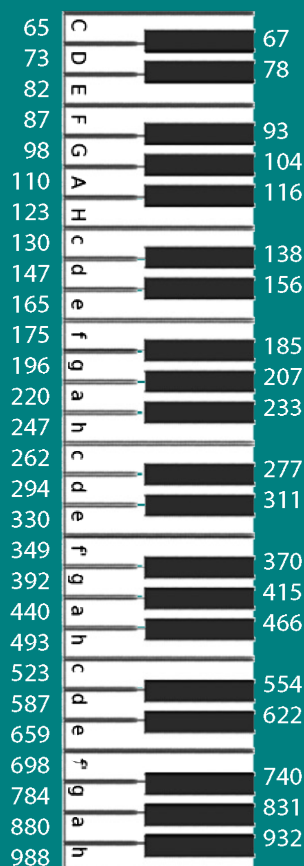
Warum delay( ) ungeschickt ist

Delay stoppt alle Prozesse und legt den µC für diese Zeit komplett lahm, eleganter geht es mit millis( ) und etwas basteln. Recherchiere wie und notiere deine Lösung irgendwo.

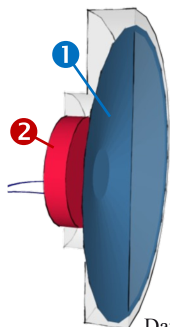
10

5 Töne, Unterprogramme, Transistor

Menschen können Schallwellen mit Frequenzen zwischen etwa 18 und 18000 **Hertz** (1 **Hz** ist eine Schwingung pro Sekunde) hören. Welche Frequenz welcher Tonhöhe entspricht, zeigt die folgende Abbildung:

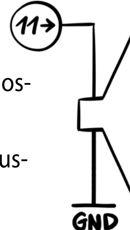


Hier lernst du, wie du mit dem Arduino und einem Lautsprecher Töne erzeugen und Tonfolgen abspielen kannst.



Viele Leute meinen, dass ein Lautsprecher sofort einen Ton von sich gebe, wenn er an eine Stromversorgung angeschlossen wird. Das ist aber falsch: Ein Lautsprecher besteht nämlich nur aus einer elastisch aufgehängten ❶ **Membran** und einem ❷ **Elektromagnet**. Fließt Elektrizität durch diesen Elektromagnet, wird die Membran angezogen, fließt keine Elektrizität, fällt die Membran wieder zurück. Ein Ton entsteht, indem die Membran schnell abwechselnd angezogen und wieder los gelassen wird, also die Stromversorgung abwechselnd ein– und ausgeschaltet wird.

Damit sich die Membran bewegt wird sie in der Mitte mit einer Spule oder einer entsprechenden Folie beklebt, die sich im Magnetfeld bewegen und die Membran mitschwingen lassen.
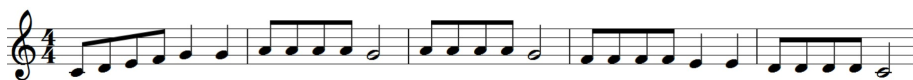


Um einen Ton zu erzeugen, kann der Lautsprecher also mit einem seiner Anschlüsse (egal welcher) zum Beispiel an Pin 11 und mit dem anderen an den Minuspol angeschlossen werden. Pin 11 muss schnell abwechselnd HIGH und LOW geschaltet werden. Weil man Töne häufig braucht, gibt es eine Anweisung, die sich um das schnelle Ein- und Ausschalten kümmert, selbst wenn der Mikrocontroller gerade etwas anderes macht:

```
void setup() {
}
void loop() {
  tone(11,440);
  delay(1000);
  noTone(11);
  delay(3000);
}
```

Diese Anweisung sorgt dafür, dass Pin 11 mit einer Frequenz von 440 Hz (Hertz) immer wieder HIGH- und LOW-geschaltet wird. Die Anweisung Tone sorgt übrigens vorher selbst dafür, dass Pin 11 in den pinMode Output geschaltet wird.

Diese Anweisung schaltet Pin 11 wieder dauerhaft auf LOW. Also ist der Ton aus.

14.1 **5.1** Schließe einen Lautsprecher an einen Pin an und programmiere ein Lied. Wenn dir kein besseres einfällt, nimm „Alle meine Entchen“:



Hinweis: Wenn zwei Töne direkt auf einander folgen, brauchst du keine noTone-Anweisung dazwischen. Es geht also auch: tone(11,262); delay(500); tone (11,294); ….

```
void setup() {  pinMode(9, OUTPUT);  }
void loop() {
  for (int i=18; i <= 25; i++){
//Riffelholz
        digitalWrite(9, HIGH);
        delay(30-i);
        digitalWrite(9, LOW);
        delay(30-i);          }

  delay(200);

  for (int i=0; i <= 20; i++){
//High-Drum
        digitalWrite(9, HIGH);
        delay(i);
        digitalWrite(9, LOW);
        delay(i);           }
}
```

Aufgaben: Töne und Funktionen

a) Bearbeite die zwei Seiten aus dem Arduino-Arbeitsheft der NwT-Fachberater gefördert durch die Gisela und Erwin Sick-Stiftung. (Ehre wem Ehre gebührt)



Statt des Lautsprechers verwenden wir am KvFG einen passiven Buzzer, dessen beide Anschlüsse direkt an PIN 11 und GND angeschlossen werden können - einfach draufstecken. Bitte + an Pin 11 und - an GND beachten!

Für die Aufgabe 15.2 dürft ihr euch auch einen großen Lautsprecher aus dem Schrank holen.

…und for-Schleife

Lernziele:
Funktionen verstehen und anwenden
Transistor als Verstärker einsetzen
for-Schleife anwenden

Falls dir dieser Stil hilft, das Programmieren besser zu verstehen, kannst du das Heft kostenlos herunterladen: nwt.schule/arduino1.pdf
Noch ein anderes Anleitungsheft: https://sfz-bw.de/miscella/arduino_script.pdf

Programmteile, wie zum Beispiel ein Refrain, die mehrfach benötigt werden, muss man nicht mehrfach programmieren. Man legt dafür ein sogenanntes Unterprogramm an, das man wie eine ganz gewöhnliche Anweisung überall dort aufruft, wo es ausgeführt werden soll.

15.1 **5.2** Vielleicht hat auch dein Musikstück einen Takt oder einen Refrain, der sich wiederholt? Probiere, diesen Teil als **Unterprogramm** zu schreiben:

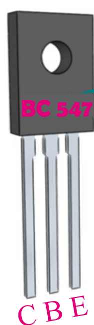
```
void setup(){        Setup
}
void loop(){         Loop
  …
  refrain();
  …
  refrain();
  …
}
void refrain(){
  tone(11,220);
  delay(300);
  …
}                    Unterprogramm
```

Am Ende fühlt sich ein Unterprogramm wie eine selbst geschriebene Anweisung an, hier zum Beispiel mit dem selbst gewählten Namen refrain(). Immer, wenn du refrain() schreibst, wird dein selbst definiertes Unterprogramm aufgerufen und ausgeführt.

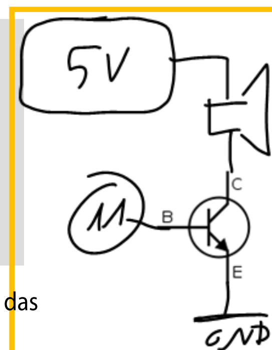
Hier siehst du, wie das Unterprogramm mit dem Namen refrain() definiert wird. Es sieht genau so aus wie setup oder loop, hat aber einen selbst gewählten Namen und natürlich einen eigenen Zweck.

Wie viele Zeilen kannst du in deinem Musikstück durch Unterprogramme sparen?

15.2 Dein Lautsprecher ist relativ leise, weil der Arduino durch seine Pins nur geringe elektrische Leistungen leiten kann. Höhere Leistung kannst du am Pin V_{IN} beziehen, das aber nicht programmierbar sondern einfach immer an ist. Mit einem **Transistor** kannst du trotzdem V_{IN} für den Lautsprecher nutzen:



Ein Transistor ist wie ein Schalter, der mit einem Pin gesteuert werden kann. Dieses Steuerpin wird an dies Basis (B) angeschlossen. Ist das Steuerpin HIGH, schließt sich der Schalter und lässt Elektrizität von C nach E durchfließen (es geht nur in dieser Richtung). Ist es LOW, fließt nichts.



Schließe Transistor und Lautsprecher so an, wie es das Schaltbild zeigt. Der Ton sollte nun lauter werden.

Namensgebung: Für die Namen von Unterprogrammen kann man beliebige selbst gewählte Worte verwenden, die keine Leerzeichen und keine Sonderzeichen (wie Umlaute oder Symbole) und als erstes Zeichen auch keine Zahl enthalten. Möglich wären also oma, opa oder hund16, unmöglich sind aber 16hund, rüpel oder a#. Dabei kommt es auf Groß – und Kleinschreibung an.

Natürlich darf man keine Bezeichner verwenden, die schon in der Programmiersprache verwendet werden.

Transistor: Die Anschlüsse des Transistors heißen Basis, **Kollektor** und **Emitter**. Der Transistor ist das technische Gerät, das am häufigsten auf der Erde hergestellt wird. Das liegt auch daran, dass alleine in einem modernen Computerprozessor mehr als 1 Milliarde Transistoren verbaut sind. Im Mikrocontroller auf dem Arduino sind es etwa 100000 - Transistoren können also sehr klein gebaut werden

Aufgabe 15.3 for-Schleife
Verwende die for-Schleife um mehrere gleiche Töne nacheinander zu spielen.

```
for (int i=1, i<10, i++)
  { ...Anweisungen... }
```

ausprobieren oder Suchmaschine nutzen FL:

Aufgabe 15.4 Bonus-Hausaufgabe
Versuche ein Lied deiner Wahl mit einem möglichst kleinen Code zu programmieren (Im Notfall auch „Alle meine Entchen“).

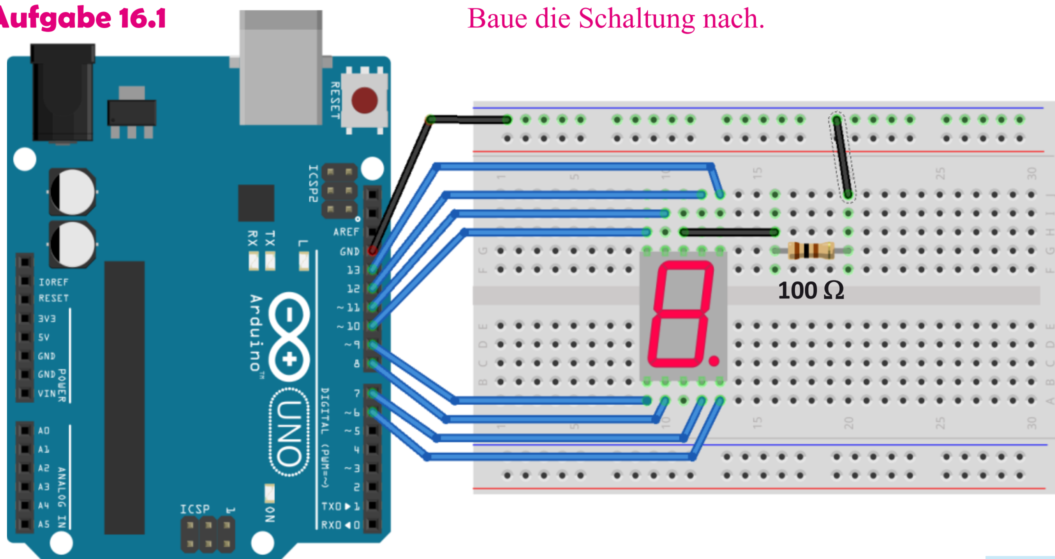
FL:

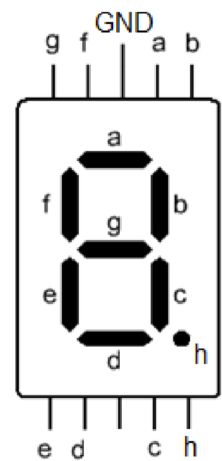
Spiel und Spaß 15.5
Der Code ganz links macht aus dem Arduino ein kleines Drum-Set. Experimentiert. Geht nur mit Lautsprecher NICHT mit Buzzer. FL:

Hausaufgabe Funktionen üben mit der 7-Segment-Anzeige

Eine 7-Segment-Anzeige (sie heißt so, weil sie aus sieben einzelnen LEDs besteht, die in einem Gehäuse verbaut sind) hat 10 Anschlüsse, die auf einer Steckplatine übersichtlich verschaltet werden können. (vgl. Abbildung). 7-Segment-Anzeigen werden z.B. in Fahrstühlen zur Anzeige der Stockwerke benutzt. Die sieben LEDs besitzen einen gemeinsamen GND-Anschluss, zwischen diesen und das GND des Arduino schalten wir den Vorwiderstand. (wenn kein 100er da ist nehmt den 220er).

Aufgabe 16.1 Baue die Schaltung nach.





Aufgabe 16.2 Definiere und teste die Anschlüsse.
Definiere `int a` bis `int h` mit den richtigen Anschlüssen und alle als `OUTPUT`. Lass alle mal leuchten und überprüfe, ob deine Definitionen stimmen.

Aufgabe 16.3 Schreibe für jede „Zahl" ein Unterprogramm.
Entweder so plump so wie im oberen Beispiel. Plump? Ja, denn die Benennung im Beispiel ist etwas aufwändig, denn du musst dann jedes Mal „Eins" tippen, du könntest die Unterprogramme auch kürzer und ebenso eindeutig u1, u2, u3, ... nennen. So wie im eleganteren Beispiel das die Notrufnummer anzeigt. Allerdings würden mit diesem kurzen Programm (mit dem analogen Unterprogramm u2 das keinen Platz mehr hatte) nach einem Durchgang alle Segmente bis auf h dauernd leuchten. Ergänze die fehlenden Befehle.

Aufgabe 16.4 Countdown und Telefonnummer
a) Programmiere einen Countdown-Zähler. Er soll in Sekundenschritten von 9 auf 0 zählen. Und dann der kleine Punkt für 2 Sekunden leuchten. FL:

b) Lass die Ziffern deiner Telefonnummer nacheinander auf der 7-Segment-Anzeige aufleuchten.
(Tipp: Verwende ein Unterprogramm um alle LEDs auszuschalten.)

Aufgabe 16.5
Schreibe Wörter wie „HALLO" und andere - so nette kleine Botschaften ;-)

FL:

Loop
```
void loop() {
Eins();
}
```

Unterprogramm
```
void Eins() {
digitalWrite(13,HIGH);
digitalWrite(7,HIGH);
}
```

Eleganteres Beispiel

Def.
```
int a=13;
int b= ... //usw.
```

Setup
```
//Setup mit OUTPUTs
```

Loop
```
void loop() {
u1();
delay(50);
u1();
delay(50);
u2();
delay(500);
}
```

Unterprogramm
```
void u1() {
digitalWrite(b,HIGH);
digitalWrite(c,HIGH);
}
```

17 Variablen

Namensgebung: Für die Namen von Variablen kann man beliebige selbst gewählte Worte mit den gleichen Einschränkungen wie bei Unterprorammen verwenden.

Speicherplatz
Der Arduino hat in seinem Speicher insgesamt etwa 2000 Byte an Platz für Variablen. Integer benötigen jeweils 2 Byte, long oder float-Variablen jeweils 4 Byte.

Überlauf
Wenn man eine zu hohe Zahl in eine Variable speichern möchte, geschieht seltsames: In einem Integer ist 32767+1 zum Beispiel –32768. Diesen sogenannten Überlauf sollte man vermeiden.

Tipp:
Man kann Variablen bereits bei der Deklaration mit einem Wert versehen. Dazu schreibt man z.B.

```
int a=8, b=-6;
long oma=87, p=99;
float haus=3.5;
```

Bei Kommazahlen schreibt man einen Dezimalpunkt.

Variablen sind Speicherplätze, in denen sich der Mikrocontroller Zahlenwerte merken kann. Dabei gibt es kleine Speicherplätze, die nur ganze Zahlen von –32768 bis 32767 speichern können, große Speicherplätze, die ganze Zahlen von -2147483648 bis 2147483647 speichern können und solche Speicherplätze, die Kommazahlen speichern können. Das folgende Beispiel zeigt, wie man diese Speicherplätze **deklariert** (mit Namen versieht), wie man Zahlen darin speichert und gespeicherte wieder benutzt. Lies es aufmerksam durch, obwohl das Programm keinen echten Sinn hat:

Def.
```
int a, b, c;
long n, m;
float x, y;
```

Setup
```
void setup(){
  a=10;
  n=100-8;
  m=2*a+15-3;
  a=200+20;
}
```

Loop
```
void loop(){
  tone(11,a);
  delay(n);
  noTone();
  delay(m);
}
```

Hinter die Anweisung int kannst du Namen von Variablen schreiben, die Zahlen von –32768 bis 32767 speichern sollen. Diese Variablen nennt man auch „**Integer**".

Hinter die Anweisung **long** schreibst du die Namen von Variablen für große Zahlen von –2147483648 bis 2147483647.

Diese Zeile macht x und y zu Variablen für Kommazahlen.

Das **Gleichheitszeichen** bedeutet, dass der Wert rechts vom Gleichheitszeichen in die Variable links davon gespeichert werden soll.

Rechts vom Gleichheitszeichen kannst du auch Rechnungen hinschreiben. Diese werden dann berechnet und das Ergebnis in der Variable gespeichert. Rechenzeichen sind +, -, * und /.

Speichert man einen neuen Wert in eine Variable, wird der alte ersetzt.

Du kannst Variablen überall verwenden, wo sonst Zahlen stehen.

17.1 Welchen Wert haben die Variablen a, n und m am Ende des oberen Beispielprogramms?

17.2 Lege in deinem Musikstück (aus Aufgabe 5.2) Variablen für die Tondauern an, z.B. a für Achtelnoten, v für Viertel, h für Halbe und benutze sie in den Delays. So kannst du die Geschwindigkeit deines Stückes sehr leicht anpassen.

17.3 Was wird das Programm rechts wohl tun?

Probiere es aus!

```
int f;
void setup(){
  f=100;
}
void loop(){
  tone(11,f);
  delay(100);
  f=f+1;
}
```

LED flackern lassen mit einem array

FL:

Ein Array (Feld) ist eine Sammlung von Werten auf die mit einer Index Nummer zugegriffen wird. Die Indexnummer fängt bei einem Array immer bei 0 an.
Ein Array mus deklariert werden und optional mit Werten belegt werden bevor es genutzt werden kann. (Datentyp der Plätze steht vor der Definition)

```
int myArray[] = {wert0, wert1, wert2...}
```

Genau so ist es möglich ein Array zuerst mit Datentyp und Größe zu deklarieren und später einer Index Position einen Wert zu geben.

```
int myArray[5];
myArray[3] = 10;
```

Array auslesen: `x = myArray[3];`

Arrays werden oft für Schleifen verwendet, bei dem der Zähler der Schleife auch als Index Position für die Werte im Array verwendet wird. Das folgende Beispiel nutzt ein Array um eine LED zum flickern zu bringen.

```
int flicker[]={180, 30, 255, 200, 10, 90, 150};
for(int i=0; i<6; i++) {
        analogWrite(ledPin, flicker[i]); delay(200); }
```

```
char Buchstabe1; //ein einzelnes Zeichen

string Wort1; //Array aus Chars

const int RoteLED = 9;//die „constante“ Variable kann im Programmablauf nicht mehr geändert werden (braucht evtl. auch weniger Speicherplatz) wird gerne für die Zuweisung der Ein- und Ausgänge verwendet.
```

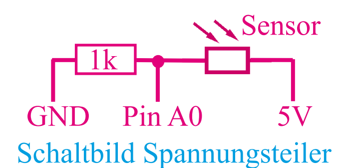
Sensoren sind Bauteile die bei variablen Umweltbedingungen variable Widerstände haben. Diese Widerstände werden vom Arduino mit Hilfe eines Spannungsteilers erfasst.

Ein einfaches Modell für Widerstände

In einem leitenden Metall sind die Elektronen (fast) alle frei beweglich. In einem Material mit Widerstand sind viele Elektronen fest gebunden und nur einige frei beweglich. Je weniger beweglich sind, desto größer ist der Widerstand.

Ein LDR ist so eine Art verkappte Solarzelle





Schaltbild Spannungsteiler

Ein einfaches Modell für variable Widerstände=Sensoren

Durch veränderte Umweltbedingen verändert sich die Zahl der freien Elektronen im Sensor. Bei einem Wärmesensor versorgt zum Beispiel die Wärmeenergie einige Elektronen mit soviel Energie, dass sie sich loslösen können und zusätzlich für den Stromfluss zur Verfügung stehen - das bedeutet, dass der Widerstand des Sensors mit zunehmender Wärme sinkt.
Bei einem Lichtsensor ist es ähnlich: Die Lichtenergie versorgt Elektronen mit so viel Energie, dass sie frei werden. Je mehr Licht auf den Sensor fällt, desto mehr Elektronen werden frei und desto stärker sinkt der Widerstand. Der Lichtsensor (LDR - light dependent resistor) reagiert sehr schnell und ist daher gut für unsere Versuche geeignet.

Erfassung des Sensorwiderstands mit einem Spannungsteiler

Wir nutzen aus, dass sich die Gesamtspannung in einer Reihenschaltung aufteilt. Zuerst messen wir den Widerstand des Sensors im Normalzustand und suchen uns dann einen ähnlichen Kohle-Widerstand aus. Jetzt werden beide in Reihe gesteckt. Dabei legen wir 5V und GND an beiden Enden der Reihe an. Die geteilte Spannung können wir jetzt messen, indem wir ein Steckkabel in die Mitte der Reihenschaltung und in A0 stecken. Der Arduino kann unterschiedliche hohe Spannungen zwischen 0 und 5 V erfassen und gibt diese als Zahlenwert aus. Da in einer Reihenschaltung von Widerständen die Spannung proportional auf die Widerstände aufgeteilt wird - ein sogenannter Spannungsteiler können wir über die Spannungsmessung den variablen Widerstand messen. Für das Einlesen der Spannung verwendet man zwei neue Befehle: Im `Setup` wird er `PinMode` auf `INPUT` gesetzt und im `Loop` wird der Spannungswert mit `analogRead` in einer Variable (die im Definitionsbereich mit `int` definiert wurde) gespeichert.

Neue Befehle Spannung messen

Def.
```
int LDRwert;
```

Setup
```
pinMode (A0, INPUT);
```

Loop
```
LDRwert = analogRead(A0);
delay (50);
```

Das kurze delay ist sinnvoll um den Spannungswert auszulesen.

Jetzt kannst du zuerst die Schaltung stecken oder gleich weiterlesen.

Ausgabe der gemessenen Werte

Für die meisten Anwendungen wäre es sehr geschickt, die gemessenen Werte auch auszugeben. Für diese Werteausgabe verwenden wir neue Befehle. Dazu eine keine Vorgeschichte:
Der Arduino wurde so konzipiert, dass er „seriell kommunizieren“ kann. Das bedeutet, dass alle Informationen (Daten) auf einer einzige Leitung eine nach der anderen - also in einer Serie - übertragen werden. Das klingt erstmal langsam, kann aber auf sehr schnelle Datenraten (Baud-Raten) eingestellt werden. Die Datenübertragung läuft über die Pins 0 und 1 die NICHT verwendet werden sollten, wenn die serielle Kommunikation verwendet wird.
Die serielle Kommunikation wird im `Setup` mit `Serial.begin`(Baudrate) gestartet und im `Loop` werden die Werte auf den „PC-Bildschirm gedruckt“ mit Serial.println(Wert). Texte in Anführungszeichen: Serial.print(“Licht =“).

Neue Befehle serielle Kommunikation

```
Serial.begin(115200);
```

```
Serial.print (“LDR: ”);
Serial.println (LDRwert);
```

Die Erweiterung „ln“ bedeutet „line“ und ergibt eine neue Zeile, ohne ln werden alle Werte hintereinander geschrieben und werden unlesbar.

analoge Sensoren

Lernziele
Aufbau und Funktion eines Spannungsteilers

Aufgabe 19.1

a) Stecke einen Spannungteiler mit LDR und Widerstand nach Schaltbild.

b) Neue Befehle stehen auf der linken Seite. Schreibe sowohl die Befehle für das Messen der Spannung als auch die für die serielle Kommunikation in ein gemeinsames Programm. Lese die Folgeseite durch um den Seriellen Monitor am PC einzuschalten und danach:

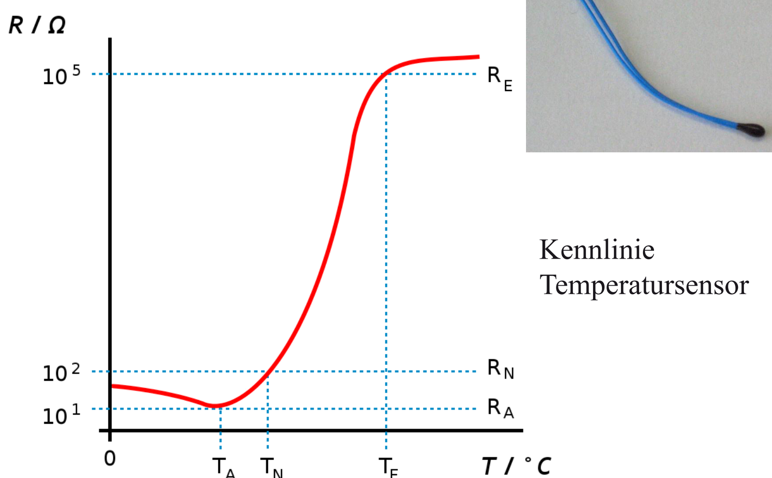
c) Notiere den typischen Wertebereich für

dunkel: von ____________ bis _______________ und

hell: von ____________ bis _______________.

Aufgabe 19.2 evtl. Hausaufgabe

Analog zum LDR kann man auch andere widerstandsverändernde Sensoren einbauen, wie diesen Temperatursensor.



Kennlinie Temperatursensor

Sensor-Kennlinien erhält man durch eigene Messreihen oder wesentlich schneller durch einen Blick in das passende Datenblatt („datasheet“).

a) Beschreibe das Diagramm (mündlich).
   1) Achsen/Einheiten/Maßstab
   2) Kurvenverlauf

b) In welchem Temperaturbereich ist der Temperatursensor sinnvoll (eindeutige Zuordnung und gute Auflösung der Widerstandswerte zu den Temperaturwerten) einsetzbar?

Aufgabe 19.3

a) Der Arduino misst die Spannung zwischen A0 und GND. Wenn Licht auf den LDR fällt ändert sich dessen Widerstand. Wie ändert sich die gemessene Spannung bei Lichteinfall? Achtung! Genau nachdenken!

FL:

b) Tausche die Reihenfolge von Sensor und Widerstand in der Spannungsteilerschaltung. Wie verändern sich die ausgebenen Werte? Beschreibe.

FL:

c) Wie verändern sich die Werte bei anderen Widerstandswerten? Wäre ein kleinerer oder ein größerer Widerstand besser geeignet als ein gleich großer? Probiere es aus und notiere das Ergebnis deiner Nachforschungen.

FL:

Serieller Monitor

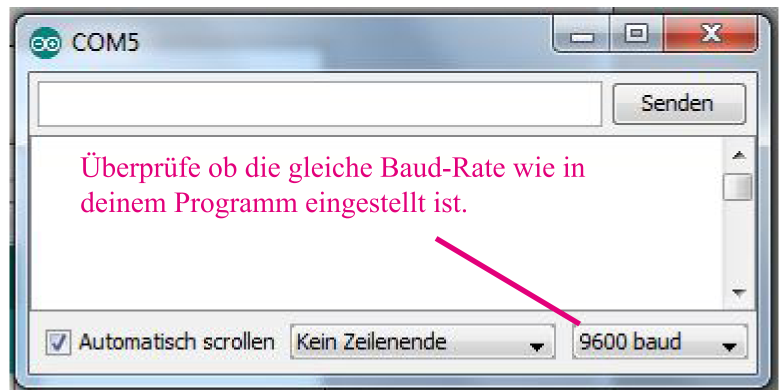
Überwachung des Programmablaufs über die serielle Schnittstelle

Wenn eine serielle Kommunikation programmiert wurde kann man den seriellen Monitor am Laptop satrten. Klicke dafür auf die Lupe rechts oben im IDE-Fenster.

Hier klicken für seriellen Monitor



Danach öffnet sich das nebenstehende Fenster. Überprüfe rechts unten die Baud-Rate.
Dann solltest du die Messwerte der Reihe nach sehen.



Manchmal „will" ein Programm einfach nicht funktionieren. Dann kann es sinnvoll sein, zu überprüfen bis zu welcher Stelle das Programm denn gut läuft. Dazu kann man die serielle Kommunikation verwenden.

Im `loop` können dann passend zu den kritischen Programmschritten durch folgende Befehlen informative Rückmeldungen an den PC ausgegeben werden (Beispiele):

```
Serial.println("Schritt 1 erreicht");

Serial.println("Dunkelheit erkannt");
```

while (!Serial) { }

In vielen Arduino Beispielprogrammen wird die serielle Kommunikation so gestartet:

```
Serial.begin(9600);
  while (!Serial) { }
```

Was macht der zweite Befehl? Während `(while)` die Kommunikation nicht tut („`!Serial`"=NOT-Serial) soll er das tun. was in der geschweiften Klammer `{ }` steht: Nichts :-)
Das Programm startet also erst, wenn die Kommunikation wirklich steht. Das kann für manche Programme ganz sinnvoll werden, vor allem dann wenn sie vom PC aus gesteuert werden sollen.

Baud-Rate

Serial.begin (115200); legt die Datenstromstärke in der Einheit baud fest. Definition: Bei der Datenstromstärke 1 baud wird 1 Symbol pro Sekunde übertragen.
Im Falle einer USB-Verbindung, die nur die Werte HIGH (5V) und LOW (0V) kennt, stimmt die baud-Rate mit der Zahl der pro Sekunde übertragenen Informationen (bit pro Sekunde) überein. Die 115200 ist in der IDE voreingestellt. Wenn etwas nicht richtig funktioniert, kann man die auch auf `9600` runtersetzen, das ist ziemlich langsam aber sie funktionieren dann auch sicher und es genügt völlig zur Übermittlung so geringer Datenmengen..

Aufgabe 20.1 Anspruchsvoll!

Finde heraus wie `call und response` funktioniert und schreibe ein Programm mit dem man entweder: (oder alle drei)
a) Die LED vom PC aus mit Tastatureingaben steuern kann (an und aus)

b) Die LED vom PC aus schneller und langsamer blinken lassen kann

c) Die LED so oft blinken lässt, wie die eingetippte Zahl vorgibt.

Auch als HA möglich!

FL:

FL:

FL:

if und else

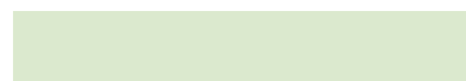
Verzweigung mit if und else

Eine erste kleine automatische Regelung soll Folgendes erledigen:
Wenn es dunkel wird, soll das Licht automatisch angehen. Wenn es hell wird, wieder aus.

Dazu wird der LDR-Helligkeitswert eingelesen und ein sinnvoller Wert festgelegt, ab dem es „dunkel“ sein soll. Wenn dieser Wert unterschritten wird, soll die LED angehen, andernfalls wird sie ausgemacht.

Für das Programmieren dieser logischen Abfolge fehlt noch die Syntax (Schreibweise) des „wennDann-andernfalls-Befehls“. Vgl. Randspalte. Beachte insbesondere, dass nach `if` und `else` keine Strickpunkte stehen. Zwischen die geschweiften Klammern können mehrere Befehle geschrieben werden. Außerdem können mehrere `if` und `else`-Konstruktionen ineinander geschachtelt werden. Das kommt später, jetzt zuerst die einfache...

Neue Befehle Verzweigung



Setup

```
if (LDRwert<111) {
       ... LED an ...
       }
else {
       ... LED aus ...
       }
```

Loop

Aufgabe 21.1

Erweitere das LDR-Programm um die Verzweigung.
Wenn du die Befehle für LED an und aus vergessen hast, kannst du auf den letzten Seiten in den Befehlsübersichten nachschauen.

Aufgabe 21.2

Damit du den Wert in der Verzweigung (if/else) richtig setzen kannst (111 ist es sicher nicht) solltest du dir die Messwerte in A0 über den seriellen Monitor ausgeben lassen und dann dein Programm entsprechend abändern oder auf Seite 17 nachlesen, welcher Wert sinnvoll sein könnte.

Aufgabe 21.3 Hausaufgabe

Schließe zuerst drei LEDs an.

Programmiere: Verwende die Befehle if und zweimal Befehl `else if` um je nach Helligkeitsstufe keine, eine, zwei oder drei LEDs leuchten zu lassen. Verknüpfe die Bedingungen (Werte zwischen 123 und 347) in den `else if` Abfragen ggfls. mit logischen Operatoren (Befehle auf der vorletzten Doppelseite oder unter arduino.cc)

Lernziel:
Befehlsübersicht im Heft kennen

Alle bekannten Befehle in einem Programm wiederholen.

Aufgabe 21.4

Schreibe ein Programm in dem die LED immer schneller (oder langsamer) blinkt je heller es wird. Tipp: Wo musst dd dann die Variable LDRwert verwenden?

FL:

Digitale Eingabe

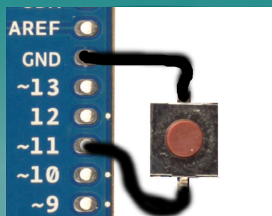
22 Eingaben mit **Tastern**

Schalter und Taster?

Der Unterschied zwischen Schaltern und Tastern ist: Bei Schaltern bleibt der elektrische Zustand nach dem Drücken beibehalten, bei Tastern nur, solange man drückt. Beide, Taster und Schalter, nennt man zusammen Schaltelemente.

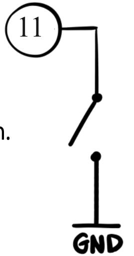
Schaltelemente mit vielen Anschlüssen?

Viele Schaltelemente haben mehr als zwei Anschlüsse. Um sie an einem Mikrocontroller betreiben zu können, musst du herausfinden, welche der Anschlüsse beim Schalter in betätigtem Zustand verbunden und in geöffneten Zustand nicht verbunden sind. Dabei kann dir ein Multimeter als Leitfähigkeitsprüfer helfen.



Am KvFG haben wir für die Anfänger die teuren Taster mit zwei Beinen die sich ganz leicht anschließen lassen

Hier lernst du, wie man an den Arduino Taster anschließt - und im Programm darauf reagieren kann, wenn sie gedrückt werden. Damit das funktioniert, muss ein Taster erstens immer zwischen einen Pin und den Minuspol angeschlossen werden. Zweitens muss der Pin nicht auf den Modus OUTPUT sondern auf INPUT_PULLUP gestellt werden.



22.1 Schließe einen Taster (oder Schalter) so an Pin **11** an, wie es das Schaltbild zeigt. Dieses Programm zeigt den Zustand des Tasters im seriellen Monitor an:

```
int i;
void setup(){
  pinMode(11,INPUT_PULLUP);
  Serial.begin(9600);
}
void loop(){
  i=digitalRead(11);
  Serial.println(i);
}
```

Diese Anweisung setzt den Pin in den Modus INPUT_PULLUP.

Hier wird der momentane Zustand des Tasters abgefragt und in der Variable i gespeichert...

...und hier dann angezeigt.

Stimmt der folgende Satz? „Ist der Taster geschlossen, wird eine 0 angezeigt, weil Pin 4 dann mit dem Minuspol verbunden ist. Ist Pin 4 nicht mit dem Minuspol verbunden, weil der Taster nicht gedrückt oder überhaupt nicht angeschlossen ist, ist das Ergebnis eine 1."

22.2 Schreibe ein Programm, das eine LED an Pin 9 ausschaltet, wenn der Taster nicht betätigt ist und einschaltet, wenn er betätigt ist. If-else hilft dabei.

22.3 Schließe zwei Taster und einen Lautsprecher an. Bei Druck auf den einen soll ein tiefer Ton gespielt werden, beim anderen ein hoher Ton.

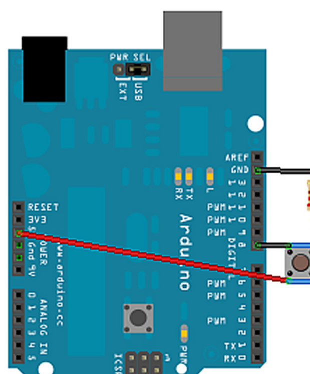
22.4 HA Schreibe ein Programm, das mitzählt, wie oft ein Taster gedrückt wird. Die Gesamtzahl soll jedes Mal auf dem seriellen Monitor ausgegeben werden.

22.5 Die Anweisung **millis()** beinhaltet immer die Anzahl der Millisekunden, die der Mikrocontroller schon läuft. Kannst du mit diesem Programm herausfinden, wie lange es etwa dauert, ein Zeichen auf den seriellen Monitor zu schreiben?

```
long m;
void setup(){
  Serial.begin(9600);
}
void loop(){
  m=millis();
  Serial.println(m);
}
```

FL:

Der auf dem Arduino befindliche AVR von Atmega besitzt 14 Eingänge. Da recht hohe Frequenzen zum Einsatz kommen (8,16 oder 20 MHz) die sich in unmittelbarer Nähe der Eingänge befinden ist es notwendig diese Eingänge mit einem definierten Potential zu versehen. Klassischer Weise wird ein Widerstand verwendet, welcher zum Beispiel auf Masse gelegt wird. Ein Pull Down Widerstand der das Potential auf 0 V runterzieht. Rechts im Bild die notwendige Beschaltung.



Da dies ein recht häufiger Einsatzfall ist, wurden im Arduino hierfür passende interne Pull-up Widerstände (20K) integriert, die sich bei Bedarf aktivieren lassen. (s.o.)
Danach ist nur noch die Außenbeschaltung von oben notwendig.
Es ist jedoch zu beachten, dass der Wirksinn nun umgekehrt ist, der Eingang hat also im Normalfall TRUE und im betätigten Fall FALSE,

Zusatzeinheit PWM

PWM (Pulsweitenmodulation) – das „analoge" als Ausgang:

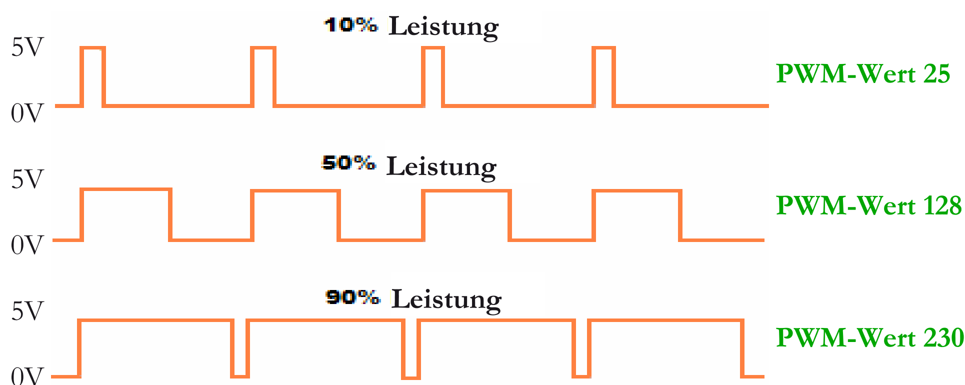
Wenn man unter einem analogen Ausgang einen solchen versteht, der alle Spannungen zwischen 0V und 5V bereitstellt, dann gibt es das nicht. Der Mikrocontroller liefert nur 0V (low) oder 5V (high).

Es gibt aber dennoch die Möglichkeit die elektrische Leistung, die ein Ausgang liefert zu verändern. Das geschieht durch schnelles Ein- und Ausschalten (Wechsel zwischen high und low), die sogenannte Pulsweitenmodulation (PWM).
Die Pins auf dem Arduino, die mit (~) versehen sind, sind in der Lage, ein PWM-Signal zu erzeugen.

Der folgende Befehl erzeugt solch ein PWM-Signal: **analogWrite(pin, Wert);**

Als **Pin** kommen nur **3, 5, 6, 9, 10, 11** in Frage, der **Wert** kann **zwischen 0 und 255** gewählt werden.

Der zeitliche Verlauf am Ausgang sieht dann wie folgt aus:



Aufgaben:

1. Schreibe ein Programm, welches die LED heller werden lässt.
   (Hinweis: Benutze dazu eine for-Schleife!)
2. Schreibe ein Programm, welches die LED je nach Helligkeit am LDR heller oder dunkler werden lässt.

*Am einfachsten löst man die Aufgabe, wenn man eine Variable **stellwert** deklariert, und den Messbereich von analogRead(0) (am Serial Monitor anzeigen lassen!) in den Stellbereich 0-255 umrechnest (Beispiel siehe rechts):* oder den `map` Befehl verwenden.

```
int stellwert = 0;
…
void loop()
{
 stellwert = analogRead(A0)/3 – 30;
 analogWrite(11, stellwert);
}
```

3. Ersetze die LED durch einen kleinen Gleichstrom-Motor mit Luftschraube. Ermittle, ab welchem Wert der Motor anläuft.

4. Baue ein Anzeigegerät mit Servo-gesteuertem Zeiger der anzeigt, wie starkt die LED leuchtet.

Obwohl die **LED** in obigem Versuch gleichmäßig heller geschaltet wird, hat man den Eindruck, dass die Beleuchtung nicht gleichmäßig zunimmt. Dies kommt daher, dass das Auge schwaches Licht weniger intensiv wahrnimmt als helles Licht. Dieser Eindruck lässt sich verhindern, wenn man zu Beginn die Beleuchtungsstärke etwas stärker macht. Dies gelingt mit einem **array**, einem Puffer für mehrere Variablen unter dem gleichen Namen. Die Deklaration erfolgt mit:

```
const int wert[5] = {23, 47, 67, 128, 255};
```

In wert[0] ist dann die Zahl 23 gespeichert, in wert[1] die Zahl 4 usw. Mit folgender Befehlskette lässt sich die LED mit diesen 5 Werten steuern:

```
for (int i = 0;i<=4;i++)
{
  analogWrite(11,wert[i]);
}
```

Schreibe ein Programm, bei dem mit Hilfe eines 32-stelligen arrays die Led so langsam angeschaltet wird, so dass der Eindruck entsteht, die LED wird gleichmäßig heller.

FL:

Entsprechendes lässt sich auch für den kleinen **Motor** mit Luftschraube durchführen. Sinnvoll wäre hier auch, den Bereich auszublenden, bei dem der Motor nicht anläuft.

FL:

Stephan Juchem, Helmut Posselt, Peter Weber 2011 Grundlagen Arduino - NTS Albstadt

Lernziele:
Taster, millis, Serial wiederholen
while-Schleife verwenden
Zufallszahlen nutzen

Wie gut ist deine Reaktionszeit?

Hausaufgabe 24.1 Baue und programmiere ein Reaktionszeitmessgerät:

Aufwändige Hausaufgabe - evtl. auf mehrere Tage verteilen...

Nachdem eine LED nach einer bestimmten Zeit zufällig (random-Befehl verwenden) aufleuchtet, soll ein Taster gedrückt und die Reaktionszeit zwischen Aufleuchten und Drücken bestimmt werden. Damit dir das nicht zu schwer fällt, hier ein Programmvorschlag aus diesem (https://sfz-bw.de/miscella/arduino_script.pdf) Arbeitsheft. (Der Link könnte dir das Tippen ersparen ;-)) Allerdings finden sich in deren Programm noch Fehler. Korrigiere sie und übernehme auch unsere Änderungswünsche!

Def / Set / Loo

```
int led = 2;
int taster = 4;

void setup() {
  pinMode(led,OUTPUT);
}

void loop() {
  int zeit = 0;     // Zeit wird auf Null gestellt
  int i = random(1000,10000);
  delay(i);
  digitalWrite(led,HIGH);
  int tasterStatus = digitalRead(taster);

  while(tasterStatus == LOW) { // Solange Schalter nicht gedrückt…
    zeit = zeit+1;
    delay(1);
    tasterStatus = digitalRead(taster);
  }
  digitalWrite(led,LOW);
  delay(5000);
}
```

Änderung: wir sparen uns das Stecken und nehmen die eingebaute LED von Pin 13 (du darfst auch eine echte LED ohne Widerstand zwischen 13 und GND stecken, denn Pin 13 hat maximal 20 mA.

Änderung: da die Variablen zeit, i und tasterStatus immer, also global benutzt werden, sollten sie auch global definiert werden (spart evtl. Speicherplatz). Im loop dann die „int"s löschen - aber nicht die Befehle dahinter!

Fehler 1: Taster im Setup vergessen. Bitte mit Pullup definieren.

Änderung: sieben Sekunden reichen

Änderung: Abfrage-Logik an Pullup anpassen!

Fehler 2 (?): Damit man random verwenden kann, muss man die Zufallszahlen eigentlich zuerst generieren. Vgl. S.34 Kasten „Zufallszahl". Oder funktioniert es auch ohne?

Änderung: drei Sekunden reichen

a) Überleg dir, in welchem Programmabschnitt oben die Zeitmessung stattfindet. Umkreise ihn.

b) Erweitere das Programm so, dass die Reaktionszeit auf dem seriellen Monitor angezeigt wird. Hier lohnt sich zum ersten Mal das `while (!Serial){}` *freu*

c) Untersuche, ob ein Mensch schneller auf optische oder auf akustische Reize reagiert. Verwende für den akustischen Reiz einen Lautsprecher, der zufällig einen kurzen Ton spielt. (Man könnte zusätzlich noch überprüfen, ob die Tonhöhe eine Rolle spielt, welche Hintergundgeräusche besonders ablenken,.... aber das führt hier zu weit - GFS in Bio?).

d) Bestimme nun für jede Variante (sehen / hören) in jeweils 20 Einzelversuchen deine Reaktionszeit. Bilde die Mittelwerte für beide Testreihen und vergleiche. Alternativ kannst du d) überspringen und gleich e) lösen.

e) Der Arduino soll die 20 Durchgänge am Stück durchführen und den Durchschnitt selbst berechnen. Tipps: (for(…,i<=20)) (Zählvariable definieren: summe = summe + zeit und am Ende durchschnitt =summe/20)

FL:

f) Erledige die Zeiterfassung mit Hilfe des millis-Befehls. Auf diese Weise kann man auf die Verwendung der while-Schleife verzichten.

FL:

Bonus Fernbedienung: Aus dem gleichen Skript wie links, also ohne Gewähr. Beachte, dass bei deinem IR-Empfänger durch den Einbau auf einer Platine die Pin-Reihenfolge vertauscht ist. Orientiere dich an der Beschriftung: + zu 5V, - zu GND, S (Signal) zu Pin 11).

Viele elektronische Geräte (Fernseher, Stereoanlage, …) lassen sich elegant mit einer Fernbedienung steuern. Die meisten Fernbedienungen arbeiten mit einer Infrarot-LED (kurz: IR-LED) als Sender, die ein codiertes Signal an den Infrarot-Empfänger sendet, der sich im Elektrogerät selbst befindet.

Aufgabe 23.1

Drücke auf einige Tasten der Fernbedienung und betrachte die IR-LED auf der Vorderseite der Fernbedienung. Betrachte anschließend die IR-LED indirekt durch eine Handykamera. Was stellst du fest?

Das folgende Programm schaltet eine LED bei Tastendruck ein und aus:

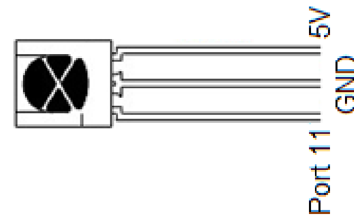
```
#include <IRremote.h>

int ir = 11;       //  Anschluss des IR-Empfängers an Port 11
IRrecv irrecv(ir);
decode_results results;

void setup() {
  Serial.begin(9600);
  pinMode(2,OUTPUT);    // LED an Port 2
  irrecv.enableIRIn(); // Empfangprozess starten
}

void loop() {
  if (irrecv.decode(&results)) {    // Falls Daten ankommen…
    Serial.println(results.value);
    if (results.value == 16724175) {
      digitalWrite(2,HIGH);
    }
    if (results.value == 16718055) {
      digitalWrite(2,LOW);
    }
    irrecv.resume();          // Empfange die nächsten Daten…
  }
}
```



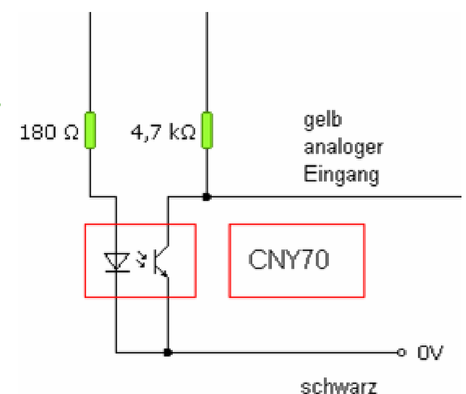


Aufgabe 23.2

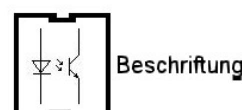
a) Finde heraus, welche Tasten der IR-Fernbedienung im Programm die LED ein- bzw. ausschalten. Halte in einer Excel-Tabelle fest, welcher Wert bei welchem Tastendruck gesendet wird.

Bonus weil noch Platz war: Als „Lichttaster" und anderes einsetzbar…

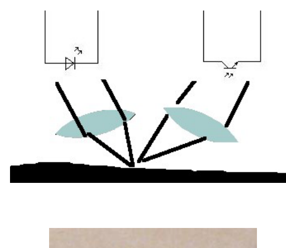
5.5 Reflexoptokoppler CNY70 (analoger Eingang)



Reflexoptokoppler können die Helligkeit von Flächen detektieren. Sie werden z.B. als Liniensensor, Radencoder, Abgrunddetektor oder Lichtschranke eingesetzt. In einem Reflexoptokoppler sind eine IR LED als Lichtquelle und ein Fototransistor als Empfänger in einem gemeinsamen Gehäuse verbaut. Der Fototransistor empfängt die von der zu untersuchenden Oberfläche gestreute IR-Strahlung und gibt ein Signal an den Mikrocontroller. Im Betrieb sollte die Entfernung zur streuenden Fläche zwischen 1mm und 4mm betragen. Allerdings ist der Sensor empfindlich gegen Streulicht, da das Licht der IR-LED nicht moduliert ist.

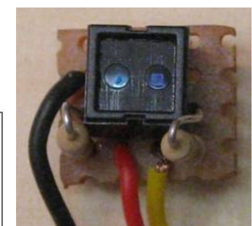


Der Strom durch den Fototransistor ist von der eingehenden Strahlungsintensität anhängig. Die Basis des Transistors wird durch den Einfall des IR-Lichtes besser leitend. Er reagiert also wie ein von der einfallenden IR-Strahlung abhängiger Widerstand.



Quelle:

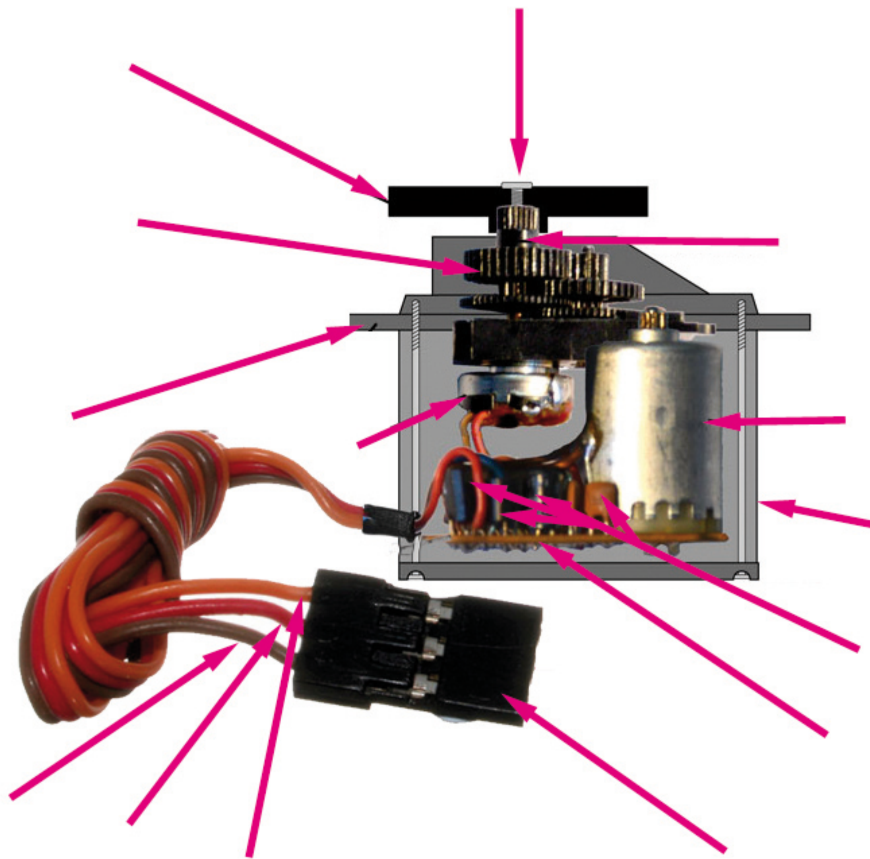




–

Servomotoren

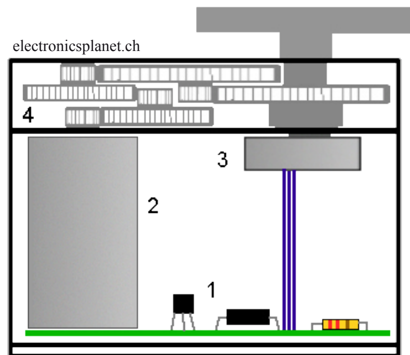
n.



Servo

Lernziele:
Eigenständige Informationsbeschaffung
Wege wissen, wie man sich sinnvoll/effektiv über unbekannte Bauteile informiert und sie effektiv einsetzt

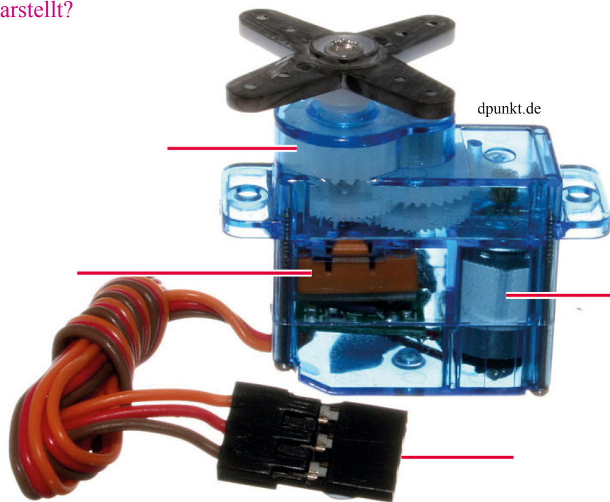
Wenn man das Poti durch einen Festwiderstand ersetzt und am Getriebe die beiden Stopp-Zapfen entfernt, kann man den Servo auf Rundlauf umstellen. Das machen wir am KvFG mit jenen Servos deren Elektronik defekt ist.



Diese Abbildungen enthalten weitere Elemente die links noch fehlen oder undeutlich dargestellt sind. Beschriften musst du bei deiesn kleinen Abbildungen nichts.



Ist dir klar, warum das Poti einen Spannungsteiler darstellt?



Folgende Begriffe tauchen in der Beschriftung des Servos links auf. Drei Pfeile haben zwei verschiedene passende Begriffe - notiere beide Begriffe an diese Pfeile. Blaue mit Funktion!

Servohorn	Anschlussstecker
Untersetzungsgetriebe	Betriebsspannung/5V
Befestigungslasche	GND/0V
Potentiometer	Gehäuse
Leiterplatte	Regelungselektronik
Antriebswelle	Schwarzes/braunes Kabel
Elektromotor	Rotes Kabel
Steuersignal	Gelbes Kabel

Aufgabe 27.1 a) ist Hausaufgabe

a) Erkläre die Funktionsweise eines Servomotors (auf der linken Seite). Konkret: Ergänze noch weitere, sinnvolle Pfeile, velängere die bestehenden und beschrifte die Abbildung (Begriffe unten links nutzen). Notiere die Funktion der wichtigen Bauteile/Bauteilgruppen (blau gedruckt) direkt in der Übersicht. Plane zuerst mit Bleistift(?).
Starte auf wikipedia.de/Servo und folge auch den ersten drei Weblinks am Ende des Artikels, falls du beim Zuhören leicht lernst, kannst du dir auch Lernvideos dazu anschauen.

b) Lerne einen Servo anzusteuern. Öffne dazu das Programm „`knob`“ (ggfls. im WWW suchen) Folge dort dem Link im Programm um zu erfahren wie du den Servo und das Poti genau anschließen musst. Bring das Ding zum Laufen.
Hilfe: dt: `Fritzing.org` engl: `arduino.cc`

c) Begründe die Notwendigkeit des `map`-Befehls im Programm „Knob.

d) Schreibe `Knob` so um, dass sich der Servo nur noch in einem 60 Grad Bereich bewegt.

FL:

e) Hausaufgabe 2:
Lernziel: Wiederholen und Vertiefen
Baue die Schaltung für „knob“ auch zuhause auf, verwende statt des Potis aber den Joystick. Wenn du ihn seitlich anschaust, erkennst du die beiden integrierten Potis. Lass die x-Achse (VRX) die Servoposition bestimmen (ausr) und die y-Achse (VRY) soll die Tonhöhe eines Buzzers festlegen (tone mit maximal delay (20)) (gut) und der integrierte Taster (SW) darf auch was cooles machen. (Keine Ahnung ob der R1 auf der Platine ein Pulldown-Widerstand ist oder ob man SW als Input_Pullup setzen muss, probier es aus!) (sgt)

FL:

Abschlussaufgabe Arduino Ampel elegant programmieren

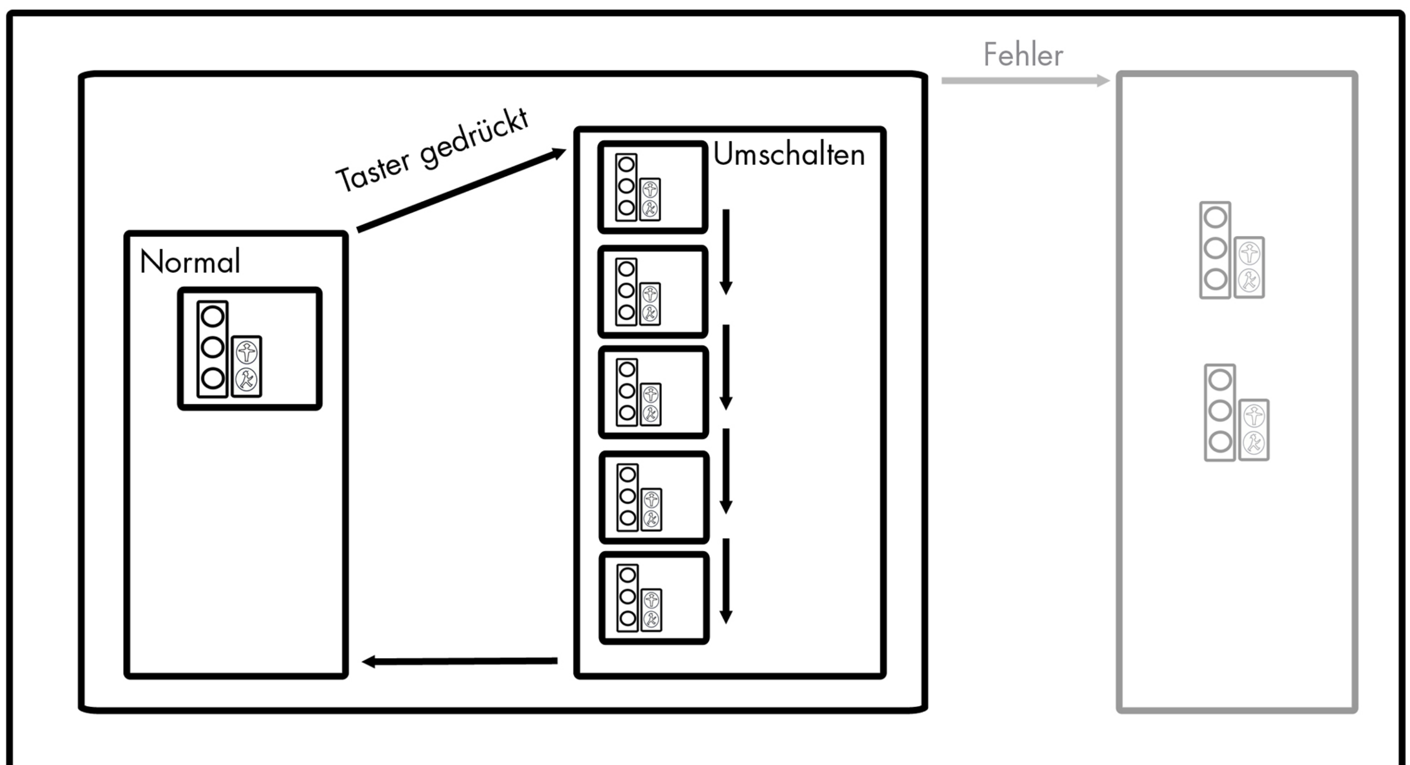
Programmiere eine Ampel
Baue die passende Schaltung auf (mit Boni?)

Um ein Programm sinnvoll schreiben zu können, sollte man sich zuerst einmal die Struktur dahinter klar machen.

Zuerst werden alle möglichen Zustände der Aktoren (bisher nur LEDs) erfasst. (Im realen Leben können Aktoren Ventile, Servos, Schrittmotoren, Lichter, Heizplatten, ... sein.)

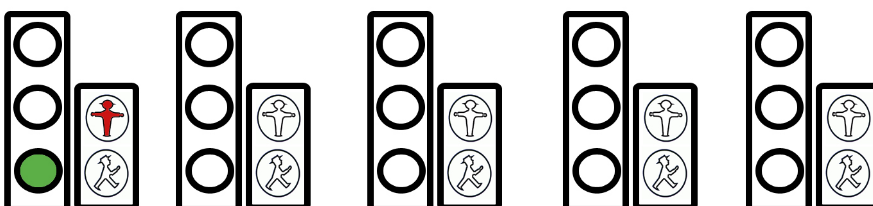
Aufgabe 28.1 Zustände einer Ampel

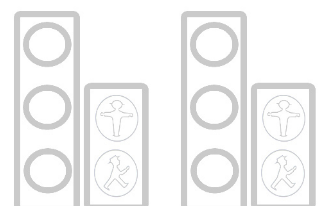
a) Bitte male in den Vorlagen jene Ampellichter die gerade leuchten bunt aus.
(Die hellgrauen Ampeln sind für die „Nacht- oder Fehler-Ampel“: Gelb blinkt, Fußgänger komplett aus).



b) Es gibt nur 5+2 verschiedene Zustände - welcher kommt doppelt vor?

c) Benenne die anderen Zustände analog zum ersten. Male sie passend an.





GRUENrot (1) ________ (2) ________ (3) ________ (4) ________ (5) ________ (6) ________ (7)

Lernziel:
Komplexes Programm selbst planen und schreiben

d) Zeichne den Pfeil ein für „Taster nicht gedrückt“.
Aufgabe 29.1: die Ampel soll normalerweise Autogrün und Fußgängerrot zeigen und nur auf Tastendruck die Fußgängerampel auf grün und wieder zurück schalten. Phasendauer 1s.

Wenn das funktioniert ist es eine ausreichende Leistung. Befriedigend ist es, wenn Code und Schaltung elegant programmiert bzw. gesteckt sind. (Code mit Unterprogrammen und Variablen. Schaltung: klare, übersichtliche Anordnung der Bauteile auf dem Steckboard, Tipps im Bild rechts beachtet).

Gut bis sehr gut wird es mit den Bonusleistungen. Damit können auch unelegante Lösungen ausgeglichen werden. Die Boni können natürlich akkumuliert werden.

Ampel funktioniert auf Tastendruck:
ausr

Code mit Unterprogr. (+0,5)
bfr-ausr
bfr

Schaltung elegant (+0,5)

Piepse-Ampel
Bonus (+0,25): Wenn der Taster gedrückt wird ertönt ein Signalton.
FL:

Lauf-schneller!-Ampel
Bonus (+0,5): Die Fußgängerampel bleibt für 1 Sekunde grün und fängt dann 10 mal immer schneller an zu blinken, bevor sie auf Rot umspringt.
FL:

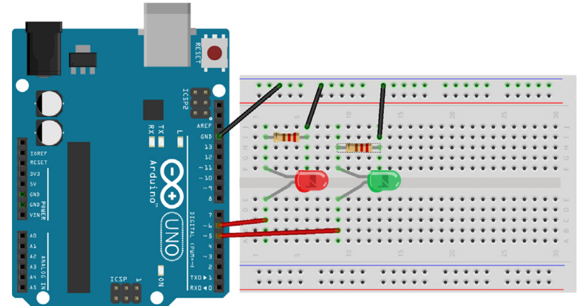
Blindenampel
Bonus (+0,5) Während die Fußgänger grün haben ertönt ein gepulster Signalton.
FL:

Wartungsmodus einfach
Bonus (+0,5): Wenn ein Schalter betätigt wird, soll die hellgraue Blinkphase eingeschaltet werden. Wie ein Schalter funktioniert? Finde es heraus!
FL:

Nachtampel
Bonus (+0,50): Wenn es dunkel wird, soll die hellgraue Blinkphase eingeschaltet werden
FL:

Wartungsmodus komplex
Bonus (+0,5): Wenn ein ein zweiter Taster gedrückt wird, soll die hellgraue Blinkphase eingeschaltet werden, so lange bis er nochmal gedrückt wird, dann soll es normal weiterlaufen.
FL:

Tipps für alle Boni für alle



Das schwarze GND-Kabel könnten wir auch direkt an den Widerstand anschließen. Wenn jedoch später mehrere Bauteile gleichzeitig verwendet werden, die alle einen GND-Anschluss benötigen, empfiehlt sich der Umweg über die Spannungsversorgungsleiste am Breadboard/ der Steckplatine. Blau = 0V/GND, Rot=5V

Unterprogramme sinnvoll einsetzen:
Prinzipieller Aufbau : Alle LEDs werden hingeschrieben und nur jene die leuchten auf HIGH gesetzt. Die auskommetierten Befehle müssen noch „richtig“ getippt werden. Jeder Zustand bekommt eine eigene Funktion. Copy and paste hilft...

```
void GRUENrot ( ) {
    digitalWrite (AutoRot, LOW);
        // AutoGelb aus;
        // AutoGruen an;
        // FussRot an;
        // FussGruen aus;
}
```

Druckknopfleuchte
Bonus (+0,25)
Sobald der Taster gedrückt wird, leuchtet eine blaue LED auf und zeigt damit an, dass gedrückt wurde. Die Ampel fängt erst eine Sekunde später an umzustellen.
FL:

Sicherheitsschranke
Bonus (+0,5) Bevor die Fußgänger Grün haben und nachdem die Autos Rot haben senkt sich eine Schranke vor den Autos. Und irgendwann sinnvoll wieder hoch.
FL:

Gesamtleistung:

Ihr bekommt zwei bis drei Projektideen zur Auswahl (die erfahrungsgemäß gut funktionieren und differenzieren). Ihr müsst darin mehrere Aktoren und Sensoren und Fischertechnik koordinieren. Jeder Aktro/Sensor der sinnvoll in den Programmablauf integriert wird gibt Bonuspunkte. Ihr könnt also fast selbst entscheiden, welche Note ihr haben wollt :-)

Jahres-Projekt - Partnerarbeit

Mindestanforderung ~ „ausr."

fischertechnik-Modell funktioniert mechanisch

fischtec:

zwei Aktoren (davon mindestens ein Motor)

Aktor 1: Aktor 2:

und ein Sensor (wenn Taster, dann zwei Stück)

Sensor 1:

koordiniert programmiert

Code fkt.:

Bonuspunktefür weitere Aktoren:

zusätzliche Motoren

Servomotor: +0,5

Servo 0,5 Servo 0,5

fischertechnik-Motor mit Transistor oder Relais +0,5

ft Motor 0,5

fischertechnik-Motor mit H-Brücke L298D +1,0

ft Motor / H-Brücke 1,0

Lüfter oder Propeller 5V +0,25

Lüfter

Schrittmotor richtig angeschlossen +1,0

Stepper 1, 0

andere

LED +0,25 für die erste, +0,1 für alle weiteren, die einzeln gesteuert werden (maximal +0,45)

LED 1 0,25 LED 2 0,1 LED 3 0,1

Extrabonus +0,25 wenn helligkeitsreguliert

reguliert

3-Farb-LED +0,5 wenn ordentlich Farbmischung genutzt

3FarbLED 0,5

Lautsprecher/passiver Buzzer +0,1 (wenn laut)

Buzz 0,1

+0,25 wenn flüsterleise

Buzzer 0,25

Piepser +0,1 weil immer laut

Piepser 0,1

Elektromagnet/Schließer +0,25

E-Magnet 0,25

Wasserpumpen u.a. mit 5V betrieben: je +0,25
mit Relais/Transistor +0,5 V

Pumpe 5V 0,25 Pumpe o.ä. mit Relais/Transistor 0,5

Bonuspunkte Programmkomplexität

Extrapunkte gibt es für komplexe Programme: für Schleifen, Unterprogramme, mehrstufige Abfrageprozesse, mehrere Aktoren sinnvoll koordiniert,...
(Wird für jede Gruppe individuell bewertet)

Komplexität

Bonuspunkte weitere Sensoren

digitale Sensoren

Taster +0,1 mit Fingerdruck maximal drei gewertet

Tast 1 Tast 2 Tast 3 je 0,1

Taster mit Hebel als Sensor für Bewegungen +0,25

Sensor 2 0,25

Reed-Kontakt mit Magnetauslöser +0,5

Reed 0,5

Lichtschranke +0,5

Lichtschranke 0,5

analoge Sensoren

LDR Licht +0,25 kalibriert +0,5

LDR 0,25 LDR kalibriert 0,5

PTC/LTC Temperatur +0,25 kalibriert +0,5

PTC 0,25 PTC kalibriert 0,5

Wägesensor +0,25 kalibriert +0,5

m 0,25 m kalibriert 0,5

Sensor auf Fertigplatine

Recherche im Netz wie man sie anschließt mit sinnvoller Integration ins Programm +0,5 (z.B. Entfernungsmesser, ...)

Entfernung 0,5 Temp/Luftfeuchte 0,5

Abstandswarner 0,5 andere 0,5

Sonderbonus

Potentiometer als Stellungssensor an fischertechnik angepasst +0,75
Alles andere - Lehrer*in fragen und hier dokumentieren!

Sonderbonus:

Mindestanforderung 3D-Druck

Mit einem 3D-gedruckten Bauteil einen Sensor oder Aktor supergut an das fischertechnik-Modell anpassen (je nach Qualität) maximal eine ganze Note Bonus. Der 3D-Druck muss dafür (fast) ohne Lehrerhilfe ablaufen. (Wäre eine Verbesserung die man gut mit zusätzlicher Zeitinvestition zu hause vorbereiten kann)

3D:

Gesamt

Braucht man oft für die speziellen Bauteile wie Sensoren auf Platinen/LCD/Servo/Schrittmotor

aus https://sfz-bw.de/miscella/arduino_script.pdf

Einbinden einer Bibliothek

Bibliotheken (sog. libraries) erweitern den Funktionsumfang der Arduino-Software um weitere Befehle. Es gibt Bibliotheken für LCDs, für Servos und viele weitere Bauteile. Will man sie verwenden, müssen sie in das Programm eingefügt werden. Die Arduino-Software hat bereits viele verschiedene Bibliotheken „an Bord“. Im Hauptmenü findet man unter Sketch den Befehl *Library importieren*. Hier wählt man einfach die Bibliothek aus, die man verwenden will und im Programm erscheint die Include-Zeile, z.B.:

`#include <IRremote.h>` (Bibliothek mit Befehlen zur Verwendung einer Fernbedingung)

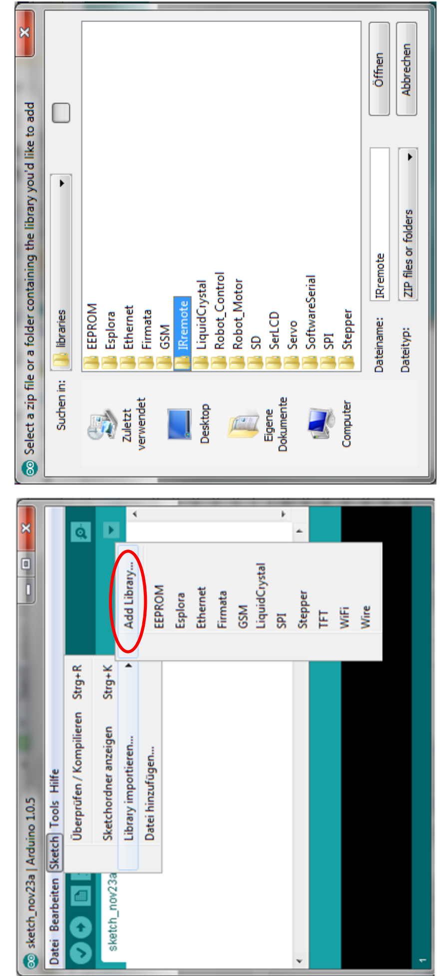
Man kann auch eine neue Library hinzufügen, die man z.B. aus dem Internet herunter geladen hat. Der entpackte Ordner der Library muss in den libraries-Ordner im Arduino-Ordner kopiert werden.

Name	Änderungsdatum	Typ	Größe
drivers	17.05.2013 23:24	Dateiordner	
examples	17.05.2013 23:26	Dateiordner	
hardware	17.05.2013 23:26	Dateiordner	
java	17.05.2013 23:26	Dateiordner	
lib	17.05.2013 23:26	Dateiordner	
libraries	22.02.2016 18:03	Dateiordner	
reference	17.05.2013 23:26	Dateiordner	
tools	17.05.2013 23:25	Dateiordner	
arduino.exe	17.05.2013 23:26	Anwendung	840 KB

In diesen Ordner wird der entpackte Ordner der Library, die eingebunden werden soll, kopiert.

Nach dem Öffnen der Arduino-Software, wird die neue Library folgendermaßen eingebunden:
Sketch → Library importieren → Add Library → gewünschten Ordner einmal anlicken → Öffnen

Anschließend kann die Library mittels *Sketch → Library importieren* eingebunden werden.



Bonus: H-Brücke anschließen aus

nwt.schule/arduino1.pdf



IC L298

Der zentrale Baustein auf der Motortreiberplatine heißt **IC** L298. IC steht für integrated circuit, also integrierter Schaltkreis. Im Inneren des L298 werkeln 8 geschickt mit einander verschaltete leistungsfähige Transistoren, eingebaut in ein einziges Gehäuse. Die Schaltung nennt man auch **H-Brücke**.

Um einen Motor zu betreiben, reichen die elektrischen Leistungen der Pins nicht aus. Und mit einem Transistor könnte man ihn zwar ein– und ausschalten, aber nicht umpolen, um seine Drehrichtung zu ändern. Deshalb verwenden wir hier eine Motortreiberplatine, mit der man zwei Motoren ansteuern kann.

An jede dieser Doppelklemmen kann ein Motor angeschlossen werden.

Über die rechten beiden Input-Pins wird den rechts angeschlossene Motor gesteuert. Ist das eine HIGH und das andere LOW, dreht er sich in eine Richtung - umgekehrt in die andere. Sind beide HIGH oder LOW, steht er still. Die linken beiden Pins sind für den linken Motoranschluss.

Dieser Stecker muss mit einer GND-Buchse des Arduino verbunden werden.

Hier wird die Versorgung für den Motor angeschlossen, also ein Netzgerät oder ein Batteriepack mit 5 bis 12 V Spannung.

15.1 Schließe an die rechte Doppelklemme einen Motor an und verbinde die rechten beiden Pins mit den Pins 3 und 4 deines Arduino. Schreibe nun ein Programm, das immer wieder Pin 3 HIGH und Pin 4 LOW schaltet, zwei Sekunden wartet, dann beide Pins HIGH schaltet und wieder zwei Sekunden wartet.

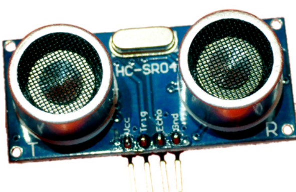
Wenn du jetzt ein Netzteil bzw. Batteriepack anschließt, sollte der Motor laufen, stehen, laufen, stehen....

15.2 Schließe nun zwei Motoren an und schreibe Unterprogramme für „beide vorwärts“, „beide rückwärts“, „links vorwärts“, „rechts vorwärts“ und „stopp“.

Mit solche Unterprogrammen bist du der Steuerung eines Fahrzeugs schon recht nahe, das zwei Motoren als Hinterräder benutzt:

Tipp!

Bei speziellen / unbekannten Bauteilen hilft es immer in Ecosia (Bäume-Pflanz-Suchmaschine) oder Google nach „Arduino -NameDesBauteils-“ (meist draufgedruckt) zu suchen. Meist kann man da gute Lösungen kopieren und in den eigenen Code einbinden.



Probleme?

Lernziel: Erst selbst prüfen, dann Lehrer*in fragen...

Start

Was funktioniert nicht?
- Leben → Schulpaner S.23
- Hardware = Bauteil oder Multimeter
- Programm / IDE

Wird es compiliert?
- nein → Fehlermeldung lesen und S. 36 = Rückseite
- ja → Wird es übertragen?

Wird es übertragen?
- ja → Programm patzt
- nein → Leuchtet die rote „ON“-Status-LED?

Upload unkt

Leuchtet die rote „ON“-Status-LED?
- nein → Steckt das USB-Kabel?
  - nein → USB-Kabel einstecken
  - ja → Kurzschluss in deiner Schaltung, schnell USB-Kabel ausstecken und überprüfen wo 5V und GND direkt verbunden sind und ändern. → Lehrer*in fragen
- ja → überprüfe die Einstellungen in der Arduino IDE unter „Tools“ >> „ serieller Port“

Werden zwei Ports angezeigt?
- nein → USB-Kabel in anderen Anschluss umstecken und 10 Sekunden warten. Neu hochladen.
- ja → Ist der Port mit USB im Namen ausgewählt?

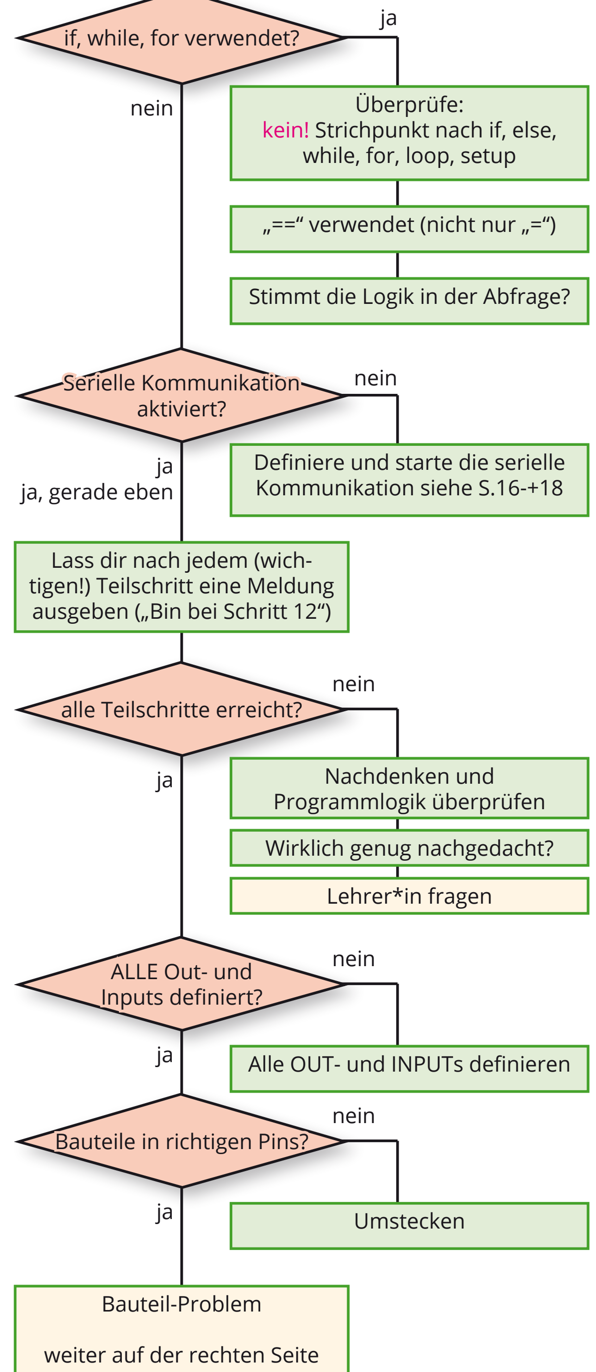
Ist der Port mit USB im Namen ausgewählt?
- nein → Port mit USB in der Bezeichnung auswählen (Haken dran?)
- ja → überprüfe die Einstellungen in der Arduino IDE unter „Tools“ >> „ Board“

„UNO R3“ ausgewählt?
- nein → „UNO R3“ auswählen
- ja → 1. Versuch?

1. Versuch?
- ja → Reset-Knopf 1 Sekunde lang gedrückt halten. Upload testen.
- nein schon der 2. Versuch → Reset-Taster drücken und gedrückt halten → Upload starten → warten bis der Upload-Balken angezeigt wird → Reset-Taster loslassen
- schon der DRITTE! → Programm speichern und Arduino IDE neu starten

Immer noch nicht?
- nö :-(→ Lehrer*in fragen → Arduino auswechseln

Programm patzt



Deutlich erweitert nach einer Idee von Stephan Juchem, Helmut Posselt, Peter Weber 2011 Grundlagen Arduino - NTS Albstadt

Multimeter motzt

- Display zeigt...
  - immer das Gleiche → Hold-Taste drücken
  - Nichts → Batterie wechseln
  - sinnlose Werte → Was gemessen?
    - Widerstand → Einstellung Ohm Ω → Ohne Batterie messen → Widerstand außerhalb der Schaltung messen
    - Spannung → Anzeige Minuswerte: Messspitzen vertauschen → Anzeige 0 V: Bauteil zwischen die Messspitzen nehmen
  - Oh, ja → Multimeter motzt

Bockiges Bauteil

Multimeter motzt → Nein →

- Widerstand → ja → Mit Multimeter messen. Achtung! Ohne Spannung/Batterie! Einstellung Ω
- Sensor wie LDR, PTC,... → ja → Mit Multimeter, ohne Spannung, bei verschiedenen Bedingungen messen (hell/dunkel, warm/kalt)
- Potentiometer → ja →
  - Drehachse auf ca. ein Drittel drehen, Widerstände zwischen je zwei Beinchen messen, alle drei sollten unterschiedlich sein - oder beim Drehen messen und prüfen ob und wie sich der Widerstandswert zwischen Schleifer- und Außenkontakt verändert.
  - Anschluss am Arduino: Außen GND und 5V, Mitte (Schleifer) an A0, (A0 als Input definieren)
- LED → ja →
  - Richtig rum gesteckt? Langes Beinchen Plus an Pin, Kante Minus an GND
  - Funktionstest am Arduino: sehr kurz zwischen 3,3 V und GND stecken
  - Anschluss mit 220 Ohm Vorwiderstand (rot rot braun)
  - Test am 9V-Block: extrem kurz, am besten mit 470 Ohm Vorwiderstand
- Infrarot-LED Fernbedienung → ja → Batterien drin? Anschließen und Handyfoto machen - leuchtet = tut doch!
- normale Diode → ja → Leitfähigkeitsmessung - darf nur in einer Richtung leiten, + und - beachten (Markierung = Minus)
- Taster → ja → Leitfähigkeit diagonal gegenüberliegender Beine testen, darf nur bei „gedrückt“ leiten. Ausmessen, welche beiden Beinchen auch „ungedrückt“ leitend verbunden sind. Sinnvoll einstecken! Setup mit INPUT-PULLUP definiert? An GND angeschlossen? Im Programm beachtet, dass bei der Definition als INPUT_PULLUP bei „gedrückt“ die „0“ = LOW als Eingang - ansonsten die „1“ = HIGH geliefert wird.
- Servo → ja → Anschlüsse: rot =+ 5V, schwarz = GND und gelb = Pin mit Tilde → Kabel ausstecken. Lässt er sich mit der Hand drehen? Nicht zu stark! Lässt er sich nur bis zu beiden Anschlagpunkten drehen?
- Kondensator → ja → Deckel gewölbt?
  - ja → Kondensator ist zerstört, Bauteil austauschen
  - nein → + und - beachtet? → entladen (Beinchen leitend verbinden) danach Leitfähigkeitsmessung - darf nicht leiten - oder Widerstand messen - sollte hoch sein
- Buzzer/Lausprecher → ja → + und - beachtet? An einem Pin mit Tilde „~“ angeschlossen? `delay` nach `tone` programmiert?
- Transistor → ja → Einfache Messung?
  - ja → Typ NPN oder PNP? Aus Datenblatt (Internet) ablesen → Transistortest am Multimeter
  - nein →
    - oft kaputt: ständig durchgesteuerte CE-Strecke. Zwischen den äußeren Beinchen Widerstand messen - er muss groß sein
    - Basis überprüfen: Ersatzschaltbild eines Transistors rechts ansehen und erkennen, das er aus 2 Dioden besteht, die an der Anode (Minuspol) verbunden sind. Dementsprechend Diodentests. Es muss das Diodenprinzip zwischen BE und BC feststellbar sein. (Leitet von Mitte nach Außen aber von Außen nach Mitte nicht). Will man einen PNP-Transistor prüfen, braucht man einfach nur
- Lehrer*in fragen

Befehlsübersicht C++ und Arduino (ganz insgesamt gibt es nur 40 verschiedene Befehle)

Variablentypen
`void` nichts
`int` natürliche Zahl
`long` große natürliche Zahl
`float` Gleitkommazahl
`char` Zeichen
`string` Text

Rechnen mit Variablen (Arithmetik)
```
x ++      // identisch mit x = x + 1, oder Erhöhung von x um +1
x --      // identisch mit x = x - 1, oder Verminderung von x um -1
x += y    // identisch mit x = x + y, oder Erhöhung von x um +y
x -= y    // identisch mit x = x - y, oder Verminderung von x um -y
x *= y    // identisch mit x = x * y, oder Multiplikation von x mit y
x /= y    // identisch mit x = x / y, oder Division von x mit y
```

Definitionen
```
int led = 13;
```
setup
```
void setup() {
  pinMode(led, OUTPUT);
}
```
loop
```
void loop() {
  digitalWrite(led, HIGH);
  delay(1000);
  digitalWrite(led, LOW);
  delay(1000);
}
```

Minimum Maximum
gilt für fast alle Datentypen, gibt den kleineren / größeren Wert zurück.
```
value = min(value, 100);
MaxWert=max(MaxWert, Messwert);
```

map-Befehl
Werte an Bereich anpassen, wenn z.B. Messwerte mw an die Servo-Ausgabe sa angepasst werden
```
sa=map(mw, 0, 1023, 0, 179);
```

Serielle Kommunikation
```
Serial.begin(9600);
while (!serial) {} //nicht nötig

Serial.print(„Text“);
Serial.println(Variablenwert);
```
Recherche: call and response

Zufallszahl
Die random Funktion erlaubt die Erzeugung der pseudo-zufälligen Werte innerhalb eines definierten Bereiches von minimum und maxium Werten. Braucht zusätzlich randomSeed als Zufallszahlenspeicher im Setup.

```
randomSeed(4); // liest die Spannungsschwankungen in Pin4 als Zufallszahlen ein. Das tut auch:
randomSeed (millis())

value = random(100, 200);
// speichert in ‚value‘ eine Zufallszahl zwischen 100 und 200
```

pins 1-13 und A0 bis A5 einlesen und ansteuern
```
pinMode (4, OUTPUT);
pinMode (LED, OUTPUT);  //Pin mit Tilde „~” s. unten
myservo.attach(9);   //Pin mit Tilde „~” nötig

pinMode (5, INPUT);
pinMode (LDR, INPUT);
pinMode (tasterpin, INPUT_PULLUP);

digitalWrite (4, HIGH);
analogWrite (LED, 112);  //Pin mit Tilde „~” nötig

a=digitalRead (5);
b=digitalRead (tasterpin);
wert=analogRead(A0);
```

C++

Servo ansteuern
```
#include <Servo.h>
Servo NameMeinesServos;

myservo.attach(9);

myservo.write(Winkelzahl von 0 -179);
delay(20);
```

Klebefläche

Kommentare

```
// Eine Zeile auskommentieren
delay(20);      // Kommentar nach Befehl
/* mehrere Zeilen
Kommentare mit Schräg-
strich Stern und am
Ende mit Stern Schräg-
strich umrahmen */
```

Logische Operatoren um mehrere Bedingungen zu verknüpfen

```
Logisch AND:
if (x > 0 && x < 5)     // nur WAHR wenn beide Ausdrücke WAHR sind
Logisch OR:
if (x > 0 || y > 0)     // WAHR wenn einer der Ausdrücke WAHR ist
Logisch NOT:
if (!x > 0)             // nur WAHR wenn der Ausdruck FALSCH ist
```

Verzweigungen

```
if (inputPin < 500){
//hier die Befehle rein
}
else{
//hier andere Befehle rein
}
else if (inputPin >= 1000){
//hier nochmal andere Befehle rein
}
```

```
switch (Variable){
  case (Wert1):
//hier die Befehle rein
  break;

  case (Wert2):
//hier andere Befehle rein
  break;

default:
//hier nochmal andere Befehle rein
break;
}
```

Variablen vergleichen mit vergleichenden Operatoren

```
x == y      // x ist gleich wie y
x != y      // x ist nicht gleich wie y
x < y       // x ist weniger als y
x > y       // x ist mehr als y
x <= y      // x ist weniger oder gleich wie y
x >= y      // x ist größer oder gleich wie y
```

Schleifen

for für definierte Anzahl an Durchläufen

```
for (int i=0; i<20; i++)
        // deklariert ‚i', teste ob weniger
{       // als 20, Erhoehung um 1
doThingA;
}
```

while - läuft nie falls Bedingung gleich erfüllt oder solange bis Bedingung erfüllt

```
while (Variable < 200)
// läuft ab solange die Variable weniger als 200 ist
{
doSomething;  // führt Anweisungen aus
Variable++;  // erhöht Variable um 1
}
```

do while - läuft mindestens ein Mal, dann weiter bis die Bedingung erfüllt ist

```
do
{
x = readSensors();
delay (50); }
while (x < 100);
// Schleife laeuft weiter bis der Messwert weniger als
100 ist
```

Funktionen / Unterprogramme

```
void setup ( ) { ...}

void loop ( )   {
      ...
      unterProgramm1 (); //Aufruf
      ergebnis = Addierfunktion (a,b);
      }

void unterProgramm1 ( ) {
                        //Definition
      ...               //außerhalb von
      }                 //loop und setup

int Addierfunktion (int x, int y) {
      int lokalesErgebnis=x+y;
      return lokalesErgebnis; }
```

Und dann gibt es noch...

I²C - Kommunikation zwischen Arduinos, Bluetooth-Kommunikation mit dem Handy, Funkmodule, Schrittmotoren, LCD-Display, Datenspeicherung auf SD-Karte, LAN-Anschluss, WLAN-Kommunikation, uvm.

Name:

Compilier-Katrastrophe

